ESCUELA SUPERIOR DE INGENIERÍA

Ingeniería Técnica en Informática de Gestión

Integración de un Motor WS-BPEL 2.0 Alternativo en el Marco de Análisis de Mutaciones MuBPEL

Curso 2013-2014

Olga Mena Gutiérrez

Cádiz, 11 de marzo de 2014

ESCUELA SUPERIOR DE INGENIERÍA

Ingeniería Técnica en Informática de Gestión

Integración de un Motor WS-BPEL 2.0 Alternativo en el Marco de Análisis de Mutaciones MuBPEL

DEPARTAMENTO: Ingeniería Informática.
DIRECTORES DEL PROYECTO: Antonia Estero Botaro y Antonio García Domínguez.
AUTORA DEL PROYECTO: Olga Mena Gutiérrez.

Cádiz, 11 de marzo de 2014

Fdo.: Olga Mena Gutiérrez

Índice general

Índice de figuras

Índice de tablas

1. Introducción

Este proyecto fin de carrera (PFC) se ha realizado en colaboración con el grupo de investigación UCASE de Ingeniería del Software, en la Universidad de Cádiz. El grupo UCASE trabaja en las líneas de investigación de *Ingeniería de servicios*, *Arquitecturas Dirigidas por Eventos*, *Arquitecturas Orientadas a Servicios*, *Desarrollo Dirigido por Modelos*, *Prueba de Software* y *Verificación y Validación de Software*.

El PFC está enmarcado en la línea de investigación de Prueba de Software de composiciones de Servicios Web o Web Services (WS) escritas en el lenguaje WS-BPEL [1]. Este lenguaje está basado en XML y permite crear procesos de negocio mediante composición de WS preexistentes y ofrecerlos a su vez como WS. El objetivo los WS es lograr la interoperabilidad entre aplicaciones mediante el uso de estándares web abiertos. La técnica de prueba que aplica el grupo a las composiciones WS-BPEL es la de prueba de mutaciones.

La prueba de mutaciones es una técnica de prueba basada en fallos. A grandes rasgos consiste generar nuevos programas (denominados mutantes) introduciendo pequeños cambios sintácticos en el programa original a probar, mediante una serie de reglas definidas para el lenguaje (operadores de mutación). Con esta técnica se consiguen dos objetivos, probar el programa y medir la calidad del conjunto de casos de prueba que se está utilizando.

El grupo de investigación UCASE ha realizado amplios avances en la aplicación de la prueba de mutaciones a composiciones WS-BPEL, habiendo definido un conjunto de operadores de mutación para este lenguaje que modelan los errores que pueden cometer los programadores [2], así como un conjunto de operadores de mutación que aplican criterios de cobertura estructural [3].

Además ha desarrollado una serie de herramientas relacionadas con esta línea. La herramienta MuBPEL [4] permite aplicar de forma automática la prueba de mutaciones a las composiciones WS-BPEL.

La herramienta GAmeraHOM [5] es el primer generador de mutantes de orden superior para composiciones WS-BPEL, que utiliza la técnica de la mutación evolutiva [6] mediante el empleo de un algoritmo genético. Esta herramienta incluye a MuBPEL que se encarga del análisis de las composiciones WS-BPEL, la generación de los mutantes y su posterior ejecución.

La herramienta Rodan [7] es un generador de casos de prueba para composiciones WS-BPEL. Este generador aplica un algoritmo genético para la obtención de casos de

prueba de calidad y utiliza la herramienta MuBPEL para la ejecución de las composiciones WS-BPEL.

1.1. Motivación

Como hemos mencionado anteriormente la herramienta MuBPEL trabaja con composiciones de servicios web escritas en el lenguaje WS-BPEL. Este lenguaje es ejecutado por un motor de ejecución WS-BPEL.

Hace unos años, cuando se empezó a desarrollar esta herramienta se seleccionó el motor ActiveBPEL 4.1 para la ejecución de las pruebas a composiciones WS-BPEL. Actualmente la empresa que desarrollaba ActiveBPEL 4.1 (originalmente Active Endpoints y ahora Informatica) ha dejado de publicar el código bajo la licencia GPL [8] después de publicar la versión 5.0, y ahora ActiveBPEL 4.1 no recibe ningún tipo de soporte. Además la documentación no es buena y la licencia GPL es incompatible a la hora de incluirlo en GAmeraHOM, cuya licencia es Apache Software License 2.0 [9] (ASL).

Por lo tanto, surge la necesidad de seleccionar un motor WS-BPEL adecuado para el grupo, e incorporarlo en aquellas herramientas que lo precisen. En este PFC nos encargaremos de ello.

1.2. Objetivos

El objetivo principal de este proyecto consiste en seleccionar e integrar un nuevo motor WS-BPEL en la herramienta MuBPEL y en todas las que hacen uso de ésta. Para conseguir este objetivo principal debemos:

- Definir una serie de requisitos, de acuerdo con las necesidades del grupo de investigación, a cumplir por el motor seleccionado.
- Comparar los motores WS-BPEL disponibles con respecto a los requisitos previamente establecidos y seleccionar el más adecuado.
- Desarrollar un protocolo de adaptación de las composiciones WS-BPEL de ActiveBPEL al motor elegido.
- Desarrollar el software necesario para la integración del motor seleccionado en MuBPEL.
- Adaptar las herramientas GAmeraHOM y Rodan de manera que utilicen el motor seleccionado para ejecutar las composiciones WS-BPEL.

1.3. Acrónimos y definiciones

A continuación se proporcionan una serie de acrónimos y definiciones que utilizaremos durante todo el proyecto para facilitar la comprensión de las explicaciones del mismo.

BPELUnit Es un framework de pruebas unitarias que hace simulaciones de la vida real, automatizado y realiza pruebas de caja blanca con las composiciones WS-BPEL.

BPTS *BPELUnit Test Suite*, conjunto de casos de prueba para BPELUnit recogidos en un fichero XML que define qué proceso se va a probar y cómo.

GAmeraHOM Sistema que genera y ejecuta automáticamente mutantes para composiciones de servicios web en WS-BPEL.

MuBPEL Es una herramienta de pruebas de mutación para el lenguaje WS-BPEL 2.0. Se puede utilizar para evaluar la calidad de un conjunto de pruebas comprobando si se puede diferenciar un mutante del programa original. Los mutantes son versiones del programa original ligeramente modificados (mutado).

Rodan Generador de casos de prueba para composiciones WS-BPEL.

WS-BPEL *Web Services Business Process Execution Language*.

XML *eXtensible Markup Language*, lenguaje de etiquetado extensible muy simple, pero estricto, que nos permite estructurar, almacenar e intercambiar una gran cantidad de datos. Las tecnologías XML son un conjunto de módulos que ofrecen servicios a las peticiones más frecuentes de los usuarios.

XPath *XML Path Language*, lenguaje que nos permite construir expresiones que recorran y procesen documentos XML. Es una idea parecida a las expresiones regulares, pero, adaptado a documentos XML.

XSLT *eXtensible Stylesheet Language Transformations*.

WSDL *Web Services Description Language*. Lenguaje en formato XML que se utiliza para describir servicios web.

GPL *GNU General Public License*. Es una licencia de software libre creada por la *Free Software Foundation*.

ASL *Apache Software License*. Es una licencia de software libre creada por la *Apache Software Foundation*.

1.4. Lenguaje WS-BPEL

WS-BPEL (*Web Services - Business Process Execution Language*) [1] es un lenguaje estandarizado por OASIS para la composición de WS. Basado en XML, representa la lógica de los procesos de negocio y los elementos que participan en ellos.

Hasta hace poco, había muchos estándares para describir procesos de negocios que competían entre sí, como el estándar Web Services Flow Language (WSFL) de IBM, XLANG de Microsoft y muchos otros. Finalmente se creó WS-BPEL (llamado BPEL4WS antes de ser estandarizado por OASIS en Abril de 2007), que combinó los mejores puntos de WSFL y XLANG. Surgió como el estándar más popular de esta categoría.

En WS-BPEL, un proceso de negocio se describe en un documento XML, siendo el entorno de ejecución del proceso de negocio. WS-BPEL permite especificar el comportamiento de un proceso de negocio basado en interacciones con WS. Este lenguaje permite a los programadores definir nuevos WS a partir de otros más simples, esto es, una composición de servicios.

La estructura de un proceso WS-BPEL se divide en cuatro secciones. Vamos a explicar cada una de ellas basándonos en el fichero BPEL que forma parte de la composición LoanApproval (LoanApproval.bpel):

1. **Definición de relaciones con los socios externos.** Los socios externos son el cliente que utiliza el proceso de negocio y los WS a los que llama el proceso.

```
1  <partnerLinks>
2    <partnerLink name="approver" xmlns:tns="http://j2ee.netbeans.org/wsdl
3    /ApprovalService" partnerLinkType="tns:approvalServicePLT"
4    partnerRole="approvalServiceRole"/>
5    <partnerLink name="assessor" xmlns:tns="http://j2ee.netbeans.org/wsdl
6    /ConcreteAssessorService" partnerLinkType="tns:riskAssessmentPLT"
7    partnerRole="riskAssessmentRole"/>
8    <partnerLink name="customer" xmlns:tns="http://j2ee.netbeans.org/wsdl
9    /ConcreteLoanService" partnerLinkType="tns:loanServicePLT"
10   myRole="loanServiceRoleType"/>
11 </partnerLinks>
```

   En este código podemos ver que participan tres WS externos, un aprobador (“approver”), un asesor (“assesor”) y un cliente (“customer”).

2. **Definición de las variables que emplea el proceso.**

```
1  <variables>
2    <variable name="risk" messageType="ns1:riskAssessmentMessage"/>
3    <variable name="approval" messageType="ns1:approvalMessage"/>
4    <variable name="request" xmlns:tns="http://j2ee.netbeans.org/wsdl
5    /loanServicePT" messageType="tns:creditInformationMessage"/>
6  </variables>
```

3. **Definición de los manejadores que utiliza el proceso.** Pueden definirse manejadores de fallos, que indican las acciones a realizar en caso de producirse un fallo interno o en un WS al que se llama. También se definen los manejadores de

eventos, que especifican las acciones a realizar en caso de que el proceso reciba una petición durante su flujo normal de ejecución.

```
<faultHandlers>
  <catch faultName="ns1:loanProcessFault" faultVariable="error"
  faultMessageType="ns1:errorMessage">
   <reply name="Reply1" partnerLink="customer" operation="request"
   portType="ns1:loanServicePT" faultName="ns1:unableToHandleRequest"
   variable="error"/>
  </catch>
</faultHandlers>
```

4. **Descripción del comportamiento del proceso de negocio.** Esto se logra a través de las actividades que proporciona el lenguaje. Las actividades pueden ser de dos tipos: básicas y estructuradas. Las básicas son las que realizan una determinada labor, mientras que las estructuradas pueden contener otras actividades y definen la lógica de negocio. A las actividades pueden asociarse un conjunto de atributos y de contenedores.

   Veamos algunas de estas actividades:

   - Actividades básicas.

     invoke Invoca una operación en un servicio web existente.

     receive Espera un mensaje de una entidad externa.

     reply Genera un mensaje para responder a una entidad externa.

     assign Permite manipular dados, como copiar un valor de una fuente a un destino.

     throw Indica un fallo interno explícitamente.

     terminate Termina la ejecución de un proceso.

     wait Permite esperar un cierto periodo de tiempo.

     empty Inserta una instrucción que no hace nada.

   - Actividades estructuradas.

     sequence Define una secuencia de acciones ordenada.

     switch Ofrece una instrucción de selección switch.

     while Define una instrucción repetitiva (bucle) while.

     pick Ofrece una instrucción de selección que funciona como el if.

flow Incluye una selección de pasos que se debe ejecutar en paralelo.

scope Define actividades anidadas.

compensate Invoca una compensación en una actividad scope que ha terminado normalmente.

```
1  <sequende>
2   <receive name="ReceiveRequest" createInstance="yes" partnerLink="customer" operation
        ="request" xmlns:tns="http://j2ee.netbeans.org/wsdl/loanServicePT" portType="tns:
        loanServicePT" variable="request"/>
3   <if name="IfSmallAmount">
4    <condition> ( $request.amount &lt;= 10000 ) </condition>
5    <sequence name="SmallAmount">
6     <invoke name="AssessRiskOfSmallAmount" partnerLink="assessor" operation="check"
          portType="ns1:riskAssessmentPT" inputVariable="request" outputVariable="risk
          "/>
7     <if name="IfLowRisk">
8      <condition> ( $risk.level = 'low' ) </condition>
9      <assign name="ApproveLowRiskSmallAmountLoans">
10      <copy>
11       <from>true()</from>
12       <to part="accept" variable="approval"/>
13      </copy>
14     </assign>
15     <else>
16      <invoke name="CheckApproverForHighRiskLowAmount" partnerLink="approver"
            operation="approve" portType="ns1:loanApprovalPT" inputVariable="request"
            outputVariable="approval"/>
17     </else>
18      </if>
19    </sequence>
20    <else>
21      <invoke name="ApproveLargeAmount" partnerLink="approver" operation="approve"
            portType="ns1:loanApprovalPT" inputVariable="request" outputVariable="
            approval"/>
22    </else>
23   </if>
24   <reply name="ReportApproval" partnerLink="customer" operation="request" portType="
        ns1:loanServicePT" variable="approval"/>
25  </sequence>
```

1.5. Servicios Web

Un servicio web es un sistema software diseñado para intercambiar datos entre aplicaciones en una red. Permite la comunicación entre aplicaciones de diferentes máquinas, ejecutadas en diferentes plataformas. Esta comunicación se consigue mediante la adopción de diversos estándares abiertos.

Un servicio Web es definido por el W3C (World Wide Web Consortium) como:

> "Un sistema software diseñado para ofrecer interacción máquina-a-máquina sobre una red. Tiene una interfaz descrita en un formato procesable por la máquina: WSDL. Otros sistemas interactúan con el servicio Web en una forma prescrita por su descripción utilizando mensajes SOAP, normalmente transmitidos usando HTTP con una serialización XML y con otros estándares Web relacionados." (W3C 2004a) [10]

La arquitectura orientada a servicios (SOA) [11] aporta importantes ventajas:

- Mayor agilidad: la reutilización de WS existentes permite crear nuevos procesos y aplicaciones de forma rápida.
- Reutilización de los servicios: añadir una nueva funcionalidad no implicar reescribir toda la aplicación, sino modificar o añadir pequeñas piezas de software manteniendo el resto.
- Reducción de costes: gracias a la reutilización se consigue reducir los plazos de implementación, simplificando también las labores de mantenimiento y soporte.

La figura 1.1 muestra la arquitectura de los WS, en la que intervienen tres elementos, el proveedor del servicio, el consumidor del servicio y el registro de servicios.

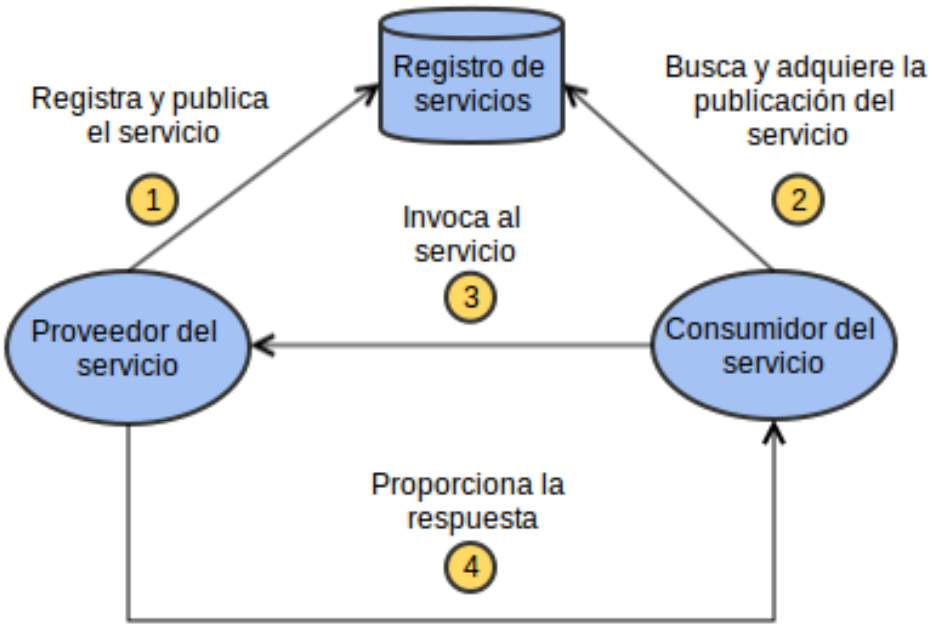

Figura 1.1.: Arquitectura de servicios Web

Proveedor del servicio Desde la perspectiva de negocio corresponde al propietario del servicio, mientras que desde la perspectiva de arquitectura se relaciona con la plataforma tecnológica que aloja el servicio.

Consumidor del servicio Desde una perspectiva del negocio es el sistema de información o aplicación que requiere satisfacer ciertas funciones empresariales. Desde la perspectiva de arquitectura es la aplicación o componente que invoca o inicia una interacción con el servicio.

Registro del servicio Corresponde a un catálogo de servicios donde se buscan las descripciones y donde los proveedores publican las descripciones. Los consumidores de servicios se conectan al directorio o registro de servicios, obteniendo información de enlace en la descripción de los mismos.

1.6. SOAP

SOAP (Protocolo Simple de Acceso a Objetos) [12] es un protocolo de intercambio de mensajes utilizado por los WS. Basado en XML, permite la interacción entre varios dispositivos y tiene la capacidad de transmitir información compleja.

SOAP especifica el formato de los mensajes. El mensaje SOAP está compuesto por un *envelope* (sobre), cuya estructura está formada por los siguientes elementos: *header* (cabecera) y *body* (cuerpo). Podemos ver una representación del mensaje SOAP en la figura 1.2.

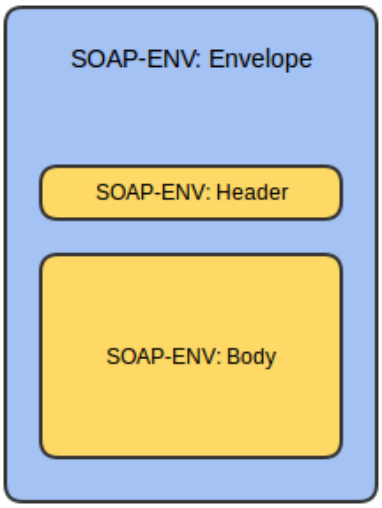

Figura 1.2.: Mensaje SOAP

Envelope Es el elemento raíz de cada mensaje SOAP.

Header Es un elemento opcional que puede contener información adicional, como información relativa a la seguridad, información de enrutamiento, etc.

Body Contiene la información principal del mensaje, que representa el cuerpo del mensaje.

Fault Representa un bloque específico contenido en el cuerpo que define un error que ocurrió durante la interacción.

1.7. XPath

XPath [13] está estandarizado por el W3C. Su propósito principal es navegar a través de elementos y atributos en un documento XML. Además de este propósito principal, también ofrece facilidad para la manipulación de cadenas, números y booleanos.

XPath modela un documento XML como un árbol de nodos. Hay diferentes tipos de nodos, incluyendo nodos elemento, nodos atributo y nodos de texto.

La sintaxis de una expresión XPath es declarativa, usando condiciones que deben cumplir los nodos resultado de dicha expresión. Se pueden filtrar nodos en función de su tipo, nombre (en caso de ser elemento), atributos, descendientes, etc.

1.8. XSLT

XSLT [14] (*XML Stylesheets Language for Transformation*, o lenguaje de transformación basado en hojas de estilo) es un lenguaje estandarizado por la W3C que permite transformar documentos XML.

Una transformación XSLT aplica reglas a una estructura de árbol leída en la entrada (un documento XML) para transformarlo en otra estructura de árbol en la salida. La transformación se logra mediante la asociación de patrones con las plantillas. Un patrón se compara con los elementos en el árbol de entrada. Una plantilla sirve para crear parte del árbol de resultados. La estructura del árbol de resultados puede ser completamente diferente de la estructura del árbol de entrada.

Una regla de plantilla XSL es un elemento `xsl:template` con un atributo `match`. Los Nodos en el árbol de entrada son comparados con los patrones de los atributos match. Cuando se encuentra un match que coincide con el nodo, los contenidos de la plantilla se representan en la salida.

Cuando se aplica una plantilla a un nodo, en principio la plantilla se aplica únicamente a éste, sustituyendo al nodo y todos sus descendientes por el resultado de la aplicación de la plantilla, lo que nos haría perder a los descendientes. Para que se

apliquen transformaciones también a los descendientes, hay que utilizar la instrucción `<xsl:apply-templates />`.

En el siguiente ejemplo se aplica la primera plantilla al nodo raíz y a continuación se aplican el resto de plantillas a los descendientes. En este caso se aplicaría la plantilla al descendiente “elemento”.

```
1  xml version="1.0" encoding="UTF-8"?>
2  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
3
4    <xsl:template match='/'>
5      <xsl:apply-templates />
6    </xsl:template>
7
8    <xsl:template match="elemento">
9    </xsl:template>
10
11 </xsl:stylesheet>
```

1.9. BPELUnit

BPELUnit [15] es una biblioteca de pruebas unitarias para composiciones WS-BPEL, creada por Philip Mayer. Puede usar cualquier motor que implemente WS-BPEL 2.0. Este framework permite a los desarrolladores probar fácilmente pequeños trozos de código que han escrito en el lenguaje WS-BPEL.

Las características principales de BPELUnit son las siguientes:

- Utiliza datos literales XML como formato de especificación de datos (BPELUnit Test Specification o BPTS), en lugar de sobres (envelopes) SOAP completos, que hace que estén al mismo nivel tanto las pruebas como las composiciones WS-BPEL.
- Permite la especificación de hilos paralelos de actividades secuenciales necesarios para la simulación de varios socios de un PUT[1] a la vez, cada uno cumpliendo un cierto protocolo de negocio.
- Integración con Apache Ant y Eclipse (figura 1.3).

Los ficheros BPTS (BPELUnit Test Specification) son ficheros XML que especifican los casos de prueba para BPELUnit. BPELUnit actuará como cliente ejecutando los casos de prueba especificados en ellos o bien si se realiza una llamada a un servicio externo, BPELUnit sustituye el servicio externo con otro servicio (*mockup*) que se comporta del modo indicado en el caso de prueba. Además BPELUnit es capaz de ejecutar peticiones síncronas y asíncronas.

Veamos la estructura general de estos ficheros. Utilizaremos un extracto del fichero LoanApproval.bpts a modo de ejemplo.

[1]Process Under Test (PUT): proceso sometido a una prueba.

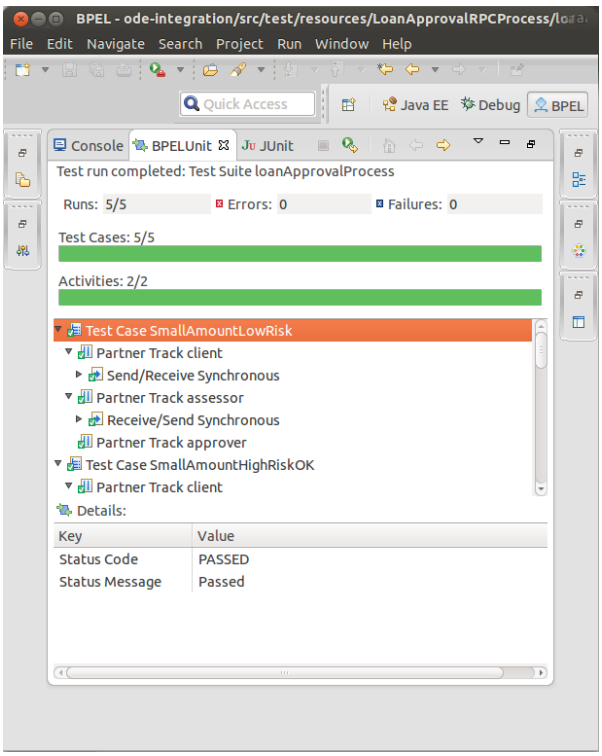

Figura 1.3.: BPELUnit ejecutándose en Eclipse

1. **Conexiones con la URL correspondiente y con los servicios WSDL.** Contiene información sobre el despliegue del proceso y los servicios WSDL que interactúan.

   Los elementos que lo componen son:

   name Especifica el nombre identificativo para el proceso.

   baseURL Determina la URL de despliegue.

   deployment Contiene información sobre el despliegue del proceso.

   - **put** Process Under Test. Representa el proceso a desplegar. El atributo `name` especifica el nombre del proceso y el atributo `type` el motor de ejecución del proceso. Este elemento contiene los ficheros WSDL de los servicios que intervienen en el proceso y define las propiedades necesarias para que la composición se despliegue.

   - **partner** Especifica los servicios externos a los que se conectará. El atributo `name` especifica el identificador del partner y el atributo `wsdl` su descripción.

El siguiente fragmento muestra la sección de despliegue de la composición *LoanApproval* para Apache ODE. Las propiedades definidas son específicas del motor.

```
1  <tes:name>loanApprovalProcess</tes:name>
2     <tes:baseURL>http://localhost:7777/ws</tes:baseURL>
3     <tes:deployment>
4        <tes:put name="loanApprovalProcess" type="ode">
5           <tes:wsdl>LoanService.wsdl</tes:wsdl>
6           <tes:property name="ODEDeploymentServiceURL">http://localhost:8080/ode/
                processes/DeploymentService</tes:property>
7           <tes:property name="DeploymentArchive">/home/olga/Documentos/ode-mubpel/src/
                mubpel/src/test/resources/LoanApprovalRPC_ODE/LoanApprovalRPCProcess</tes:
                property>
8        </tes:put>
9        <tes:partner name="approver" wsdl="ApprovalService.wsdl"/>
10       <tes:partner name="assessor" wsdl="AssessorService.wsdl"/>
11    </tes:deployment>
```

2. **Descripción de los casos de prueba.** Se encuentran recogidos en la actividad `TestCases` y puede haber desde una sola actividad, hasta varias. Cada caso de prueba se encuentra recogido en la actividad `TestCase`.

   Los elementos que lo componen son:

   testCase Define un caso de prueba. Este elemento se compone de una serie de atributos. El atributo `name` especifica el nombre identificativo del caso de prueba, el atributo `basedOn` indica si el caso de prueba se basa en uno anterior, el siguiente atributo es `abstract` que indica a BPELUnit si puede ejecutar el proceso (“false”) o no (“true”), por último el atributo `vary` especifica si se debe ejecutar más de una vez el caso de prueba cambiando los tiempos de respuesta.

   clientTrack Define la petición que se enviará desde el cliente simulado. Sólamente puede haber un cliente por caso de prueba.

   partnerTrack Define un “mockup” o servicio, y está compuesto por las mismas actividades que el `ClientTrack`. Puede haber más de uno dentro del mismo caso de prueba y cada uno con sus actividades y funciones correspondientes.

   Estos dos elementos a su vez pueden contener una secuencia de actividades que describen la interacción que se realiza en el proceso:

   sendOnly Envía un mensaje asíncrono.

   receiveOnly Espera y recibe un mensaje asíncrono.

   sendReceive Envía un mensaje y espera una respuesta en la misma conexión HTTP.

   receiveSend Espera y recibe un mensaje y envía una respuesta en la misma conexión HTTP.

sendReceiveAsynchronous Envía un mensaje asíncrono y luego espera y recibe la respuesta asíncrona.

receiveSendAsynchronous Espera y recibe un mensaje asíncrono, y luego crea y envía un mensaje de respuesta asincróno.

La mensajería asíncrona requiere ciertos campos en el encabezado SOAP para contener los ID de mensaje, responder a las direcciones u otros mecanismos que permiten devoluciones de llamada asincrónicas.

El siguiente fragmento de código muestra un caso de prueba definido para la composición *LoanApproval* de Apache ODE. Podemos observar algunos de los elementos anteriormente descritos en él.

```
1   <tes:testCases>
2      <tes:testCase name="SmallAmountLowRisk" basedOn="" abstract="false" vary="false">
3         <tes:clientTrack>
4            <tes:sendReceive service="sp:LoanService" port="LoanServicePort" operation
                 ="request">
5               <tes:send fault="false">
6                  <tes:data>
7                     <name>B A</name>
8                     <firstName>B</firstName>
9                     <amount>1500</amount>
10                 </tes:data>
11              </tes:send>
12              <tes:receive fault="false">
13                 <tes:condition>
14                    <tes:expression>accept</tes:expression>
15                    <tes:value>'true'</tes:value>
16                 </tes:condition>
17              </tes:receive>
18           </tes:sendReceive>
19        </tes:clientTrack>
20        <tes:partnerTrack name="assessor">
21           <tes:receiveSend service="as:RiskAssessmentService" port="
                 RiskAssessmentPort" operation="check">
22              <tes:receive fault="false"/>
23              <tes:send fault="false">
24                 <tes:data>
25                    <gen:level>low</gen:level>
26                 </tes:data>
27              </tes:send>
28           </tes:receiveSend>
29        </tes:partnerTrack>
30        <tes:partnerTrack name="approver"/>
31     </tes:testCase>
32
33     ...
34
35  </tes:testCases>
```

En este caso de prueba el cliente desea obtener un préstamo de la cantidad de 1500 unidades monetarias (línea 9). El aprobador comprueba si existe riesgo a la hora de conceder el préstamo. Finalmente determina que el riesgo es bajo (línea 25) por lo que se concede el préstamo al cliente (líneas 14 y 15).

2. Desarrollo y calendario

A continuación se especifica la planificación del proyecto. Describiremos la metodología empleada en su desarrollo y se enumeran las distintas tareas que se han llevado a cabo en este PFC. Estas tareas se representan mediante bloques. Algunas de éstas se pueden solapar en el tiempo y, en algunas ocasiones, no es necesario que termine una actividad para comenzar la siguiente. Además, podemos ver un diagrama de Gantt donde se muestra el desarrollo de las actividades a lo largo del tiempo. De este diagrama obtenemos los costes temporales de estas tareas en una tabla.

Este proyecto se ha realizado en 20 meses. Se empezó en Julio de 2012 estableciendo los requisitos para poder iniciarlo. Durante todo el proyecto se ha compaginado su desarrollo con la vida profesional, habiendo estado trabajando primero de becaria y después a jornada completa por lo tanto, la dedicación hacia el proyecto ha sido parcial.

A su vez, durante el desarrollo del mismo, se han realizado cuatro ponencias en los seminarios de investigación realizados por el grupo UCASE, en las que se explican los avances conseguidos en el proyecto.

2.1. Metodología

Este proyecto se ha llevado a cabo mediante una combinación de la metodología lineal (o metodología en cascada) y la metodología iterativa (o metodología incremental), aplicando el modelo de ciclo de vida Incremental. En la figura 2.1 podemos ver el esquema del modelo de ciclo de vida Incremental. Está basado en varias iteraciones lineales, de manera que al terminar cada iteración obtenemos un software útil. Pero en este caso se diferencia de la metodología lineal en que por cada iteración se revisa y se mejora el software, obteniendo una versión más estable del mismo, de más calidad, y añadiendo además nuevas funcionalidades respecto a versiones anteriores. En la última iteración obtendremos el producto final.

El modelo de ciclo de vida Incremental es el más adecuado para desarrollar este proyecto de integración ya que en primer lugar hemos de desarrollar un software básico capaz de ejecutar pruebas a composiciones WS-BPEL. Una vez realizada esta

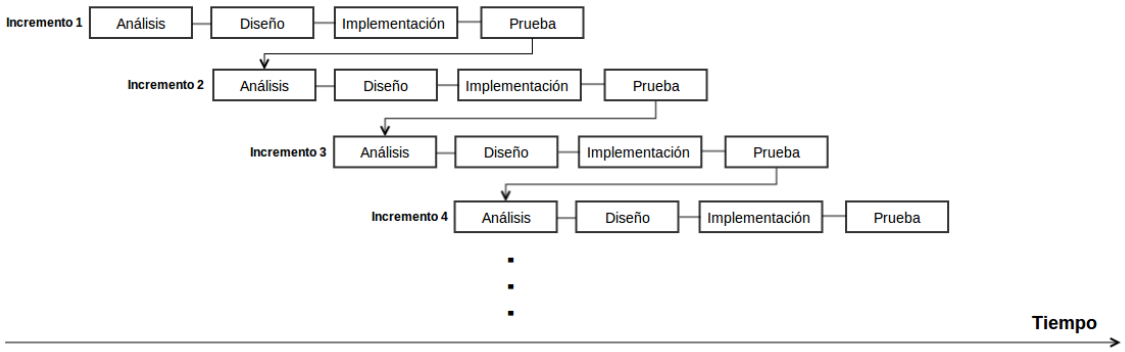

Figura 2.1.: Modelo incremental para el ciclo de vida del software

primera iteración se iniciará la integración en las herramientas de manera que, mediante sucesivas iteraciones, mejoraremos el software añadiéndole nuevas funcionalidades según se requiera, generando nuevas versiones del mismo.

2.2. Fases

A continuación se presentan las diferentes fases que se han realizado para desarrollar el presente PFC.

2.2.1. Fase 1: Establecimiento de requisitos

En este tiempo el grupo de investigación UCASE explica la necesidad de realizar este proyecto y su situación dentro de las líneas de investigación en las que se trabajan.

Además, mediante distintas reuniones entre el alumno y los profesores del grupo de investigación se establecen los requisitos necesarios para la selección del motor, detallando qué requisitos son estrictamente obligatorios y qué requisitos son opcionales para que el motor seleccionado se adecúe perfectamente a las necesidades de los proyectos del grupo.

2.2.2. Fase 2: Periodo de investigación: Estudio de los motores WS-BPEL 2.0

Una vez establecidos todos los requisitos a cumplir por el nuevo motor, se realiza un estudio de investigación sobre los diferentes motores WS-BPEL que existen hasta el momento. Para empezar, se realiza una búsqueda de todos ellos seleccionando aquellos que no están obsoletos. A continuación, se estudia cada motor por separado,

comprobando las ventajas e inconvenientes de éstos tanto de instalación como de uso, basándose en los requisitos previamente establecidos. Finalmente se selecciona el más adecuado de ellos.

2.2.3. Fase 3: Estudio de las composiciones

En esta fase seleccionamos una serie de composiciones del repositorio del grupo de investigación que posteriormente serán adaptadas para funcionar con Apache ODE. Aprendo cómo es la organización y el funcionamiento del código WS-BPEL, BPTS y WSDL. También aprendo la funcionalidad de BPELUnit realizando una serie de pruebas unitarias con estas composiciones. Además se realiza una composición directamente para Apache ODE de manera que se estudian sus ficheros específicos.

2.2.4. Fase 4: Desarrollo del protocolo de adaptación de las composiciones

Una vez estudiadas las composiciones se adaptan para funcionar con Apache ODE. De estas adaptaciones se extraen las conclusiones necesarias para desarrollar un protocolo de adaptación de composiciones WS-BPEL de ActiveBPEL a Apache ODE.

2.2.5. Fase 5 Desarrollo de ODE-integration

Se hace necesario desarrollar una biblioteca denominada ODE-integration que automatizará todo el proceso de descarga e instalación del motor y trabajará con BPELUnit para ejecutar pruebas de mutaciones a composiciones WS-BPEL.

En primer lugar se realiza una primera versión, donde se obtiene la funcionalidad básica de la misma. A continuación se ira mejorando las funcionalidades de esta biblioteca según las exigencias que requiera la integración con MuBPEL.

2.2.6. Fase 6: Integración del motor Apache ODE en MuBPEL

Para empezar fue necesario estudiar la herramienta MuBPEL. Conocer cómo funciona el código desde dentro. Una vez comprendida la funcionalidad del código de MuBPEL se integró la biblioteca ODE-integration para ejecutar las composiciones con Apache ODE desde MuBPEL con todas sus opciones.

2.2.7. Fase 7: Adaptación de la herramienta GAmeraHOM

Primeramente se realiza un estudio para conocer cómo funciona la herramienta GAmeraHOM. Posteriormente se adapta la herramienta para que ejecute las composiciones WS-BPEL con Apache ODE.

2.2.8. Fase 8: Adaptación de la herramienta Rodan

Por último se realiza la adaptación de la herramienta Rodan. Al igual que las herramientas anteriores, se realiza una primera fase de estudio para comprender cómo funciona y a continuación se realiza la fase de adaptación de la herramienta para que ejecute las composiciones WS-BPEL con Apache ODE.

2.3. Seminarios

Durante el curso he asistido a seminarios de investigación del grupo UCASE, reuniones preestablecidas en las que los distintos miembros del grupo presentaban los avances en sus determinados proyectos.

Así mismo, a medida que iba desarrollando mi proyecto, realicé cuatro ponencias sobre los avances de éste:

Seminario: 25 de septiembre de 2012 Comparativas de motores WS-BPEL. En este seminario expuse el trabajo ejercido para seleccionar el motor WS-BPEL 2.0 adecuado (Apache ODE 1.3.5).

Seminario: 10 de enero de 2013 Adaptación de composiciones web. En este seminario expliqué los pasos para transformar una composición web desarrollada para ActiveBPEL a una composición para Apache ODE, desde la instalación del motor Apache ODE, mostrando a su vez las diferencias entre ambos motores, hasta los problemas mayores obtenidos en el proceso, incluyendo un ejemplo de transformación con las composiciones LoanApprovalRPC y LoanApprovalDoc.

Seminario: 02 de mayo de 2013 Automatización de la ejecución. En este seminario describí los diferentes pasos realizados para ejecutar las composiciones web (anteriormente transformadas) de forma automática, tanto la gestión del servidor Tomcat y el motor Apache ODE como las ejecuciones de las pruebas unitarias con BPELUnit.

Seminario: 20 de enero de 2014 Integración de Apache ODE en MuBPEL. En este seminario se realiza un repaso de todo el trabajo y profundamente se explica cómo se ha integrado Apache ODE mediante la biblioteca ODE-integration en MuBPEL.

2.4. Diagrama de Gantt

El diagrama de Gantt es una herramienta que permite al usuario modelar la planificación de las tareas necesarias para la realización de un proyecto. En la figura 2.1 podemos observar el diagrama de Gantt de este proyecto en el que vemos la distribución y realización de las diferentes fases del proyecto a lo largo del tiempo.

Se ha empleado la herramienta Gantt Project para dibujar el diagrama. Está hecha en Java y disponible en `http://ganttproject.sourceforge.net`.

A continuación podemos ver la tabla 2.2 que muestra el resumen del coste temporal de las diferentes fases.

Tabla 2.1.: Diagrama de Gantt

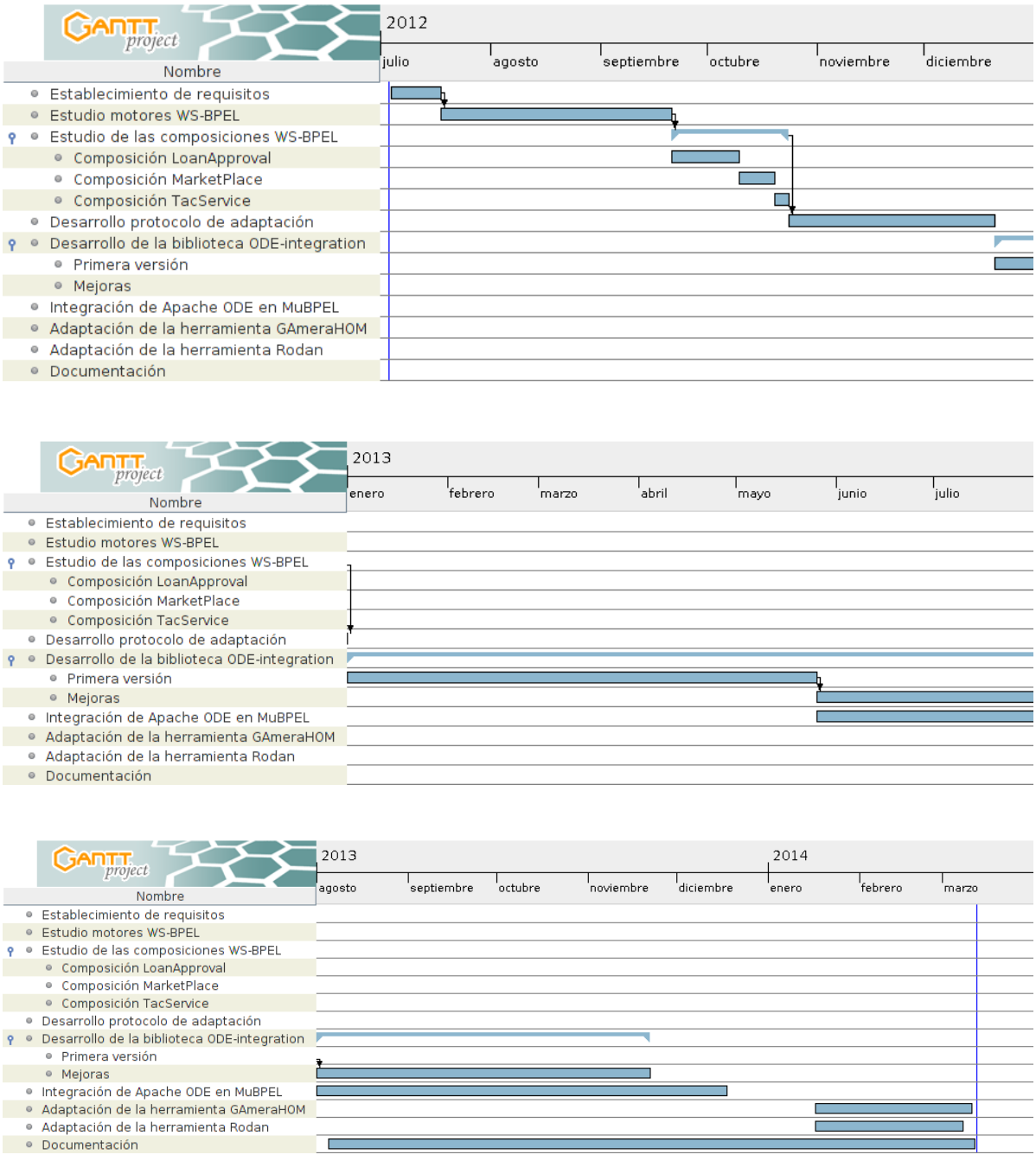

Tabla 2.2.: Fases del proyecto

Fase	Fecha inicio	Fecha fin
Establecimiento de requisitos	04/07/2012	17/07/2012
Estudio motores WS-BPEL	18/07/2012	20/09/2012
Estudio de las composiciones WS-BPEL	21/09/2012	23/10/2012
- LoanApproval	21/09/2012	09/10/2012
- MarketPlace	10/10/2012	19/10/2012
- TacService	20/10/2012	23/10/2012
Desarrollo del protocolo de adaptación	24/10/2012	20/12/2012
Desarrollo de la biblioteca ODE-integration	21/12/2012	21/11/2013
- Primera versión	21/12/2012	25/05/2013
- Mejoras	26/05/2013	21/11/2013
Integración de Apache ODE en MuBPEL	26/05/2013	17/12/2013
Adaptación de la herramienta GAmeraHOM	17/01/2014	10/03/2014
Adaptación de la herramienta Rodan	17/01/2014	07/03/2014
Documentación	05/08/2013	11/03/2014

3. Estudio de los motores WS-BPEL

Para poder seleccionar un motor WS-BPEL 2.0 adecuado a las necesidades del grupo UCASE, es necesario realizar un estudio de investigación exhaustivo sobre los motores WS-BPEL existentes hasta el momento. Se compararán todos los motores objeto de estudio hasta obtener el más adecuado. Previamente, mediante una serie de reuniones con este grupo, se establecerán los requisitos que ha de cumplir el motor seleccionado, que reflejan las necesidades del mismo. En esta sección se describen los requisitos establecidos así como el propio estudio de investigación realizado.

3.1. Establecimiento de los requisitos

En primer lugar, para poder hacer un estudio correcto de los motores WS-BPEL existentes necesitamos establecer unos requisitos previos de acuerdo con las necesidades del grupo de investigación.

Dividimos los requisitos según su grado de utilidad entre obligatorios (habrá que descartar el motor si no se cumple) y opcionales (si cumple este requisito mejor, pero no es indispensable).

A su vez, reunimos los requisitos en 4 grandes grupos:

Aspectos del proyecto en general Esta sección comprende los requisitos que ha de cumplir el proyecto desde una perspectiva global.

Aspectos de implementación Requisitos estrictamente de implementación como el lenguaje de desarrollo, compatibilidades e integraciones con otros proyectos, etc.

Aspectos de uso Este grupo contiene los requisitos sobre las funcionalidades que ofrece el motor y el uso de las mismas.

Aspectos de tecnologías SOA Requisitos relacionados con el marco de trabajo conceptual para procesos de negocios SOA.

3.1.1. Aspectos del proyecto en general

El grupo de investigación trabaja con software libre, de manera que el motor que seleccionemos debe ser un software libre. Además, uno de los problemas que hacen necesario realizar este proyecto es la incompatibilidad entre licencias, por lo tanto es importante tener en cuenta el tipo de licencia que tenga este software (recordemos que se va a integrar en un proyecto cuya licencia es ASL y por lo tanto ha de ser compatible con ésta). Un último detalle a tener en cuenta acerca del proyecto es que debe de ofrecer un seguimiento activo del mismo, ya que en algún momento será necesario resolver dudas y problemas. En consecuencia, los requisitos establecidos con respecto al proyecto en general son los siguientes:

Código abierto El código debe estar libremente disponible, sin pagar nada ni registrarse. (Obligatorio).

Licencia del código Debe ser una licencia reconocida por la Open Source Initiative. (Obligatorio).

Compatibilidad con ASL 2.0 La licencia debe ser compatible con la Apache Software License 2.0. (Obligatorio).

En desarrollo activo La última revisión subida al repositorio de código fuente debe ser como mucho de hace un año. No queremos proyectos estancados o abandonados. (Obligatorio).

Soporte de la comunidad Debe tener una lista de correo o un foro en el que preguntar dudas y problemas, y tienen que estar relativamente activos (el último mensaje debe ser de hace un año como mucho). (Obligatorio).

3.1.2. Aspectos de implementación

Los proyectos en los que vamos a integrar el motor WS-BPEL se ejecutan mediante el JRE [16] (*Java Runtime Environment*), por lo tanto nuestro motor debe estar desarrollado en un lenguaje compatible con el JRE y la plataforma OpenJDK [17]. El sistema operativo que utilizamos en el grupo es Linux. Además sería interesante que el motor fuese compatible con Maven [18] ya que esta herramienta nos facilita la construcción e integración del software en los demás proyectos. Así pues, los requisitos establecidos en esta sección son:

Lenguaje de implementación Debe estar escrito con lenguajes compatibles con el Java Runtime Environment. (Obligatorio).

Compatibilidad con Linux y OpenJDK Debe correr bajo Ubuntu 10.04 LTS con el JRE OpenJDK 6. (Obligatorio).

Compilable El código tiene que poderse compilar e instalar bien. (Obligatorio).

Integración con Maven (Opcional).

3.1.3. Aspectos de uso

El motor WS-BPEL lo usaremos para ejecutar composiciones de servicios web escritas en el lenguaje WS-BPEL. Por esta razón es imprescindible que el motor sea capaz de desplegar las composiciones. Otro detalle a tener en cuenta es que su manejo sea sencillo, tanto la instalación como la administración. Además las herramientas en las que vamos a integrar el motor utilizan BPELUnit para realizar pruebas unitarias a estas composiciones por lo tanto sería interesante que este motor tenga integración con BPELUnit. Por último, para ayudar a la integración del motor en las demás herramientas es necesario que sea compatible con algún IDE. Por consiguiente los requisitos constituidos son:

Servicios web de despliegue El motor debería permitir desplegar y/o retirar una composición subiendo un fichero a una dirección oportuna. (Obligatorio).

Facilidad de instalación La instalación básica debería consistir en descomprimir un pequeño conjunto de ficheros en el sitio oportuno. (Opcional).

Interfaz web de administración El motor debería tener una interfaz web para gestionar las instancias de los procesos, ver por dónde van, desplegar/retirar procesos, ver el registro de errores, y configurar el motor. (Opcional).

Integración con IDE Si tiene integración con Eclipse o NetBeans, mucho mejor. (Opcional).

Integración con BPELUnit (Opcional).

Estabilidad El motor debería ser razonablemente fiable y estable. (Opcional).

3.1.4. Aspectos de tecnologías SOA

Los WS se implementan utilizando ciertas tecnologías SOA como WS-BPEL 2.0 o WSDL 1.1, por lo tanto el motor seleccionado debe soportar estas tecnologías. Además sería importante que las composiciones WS-BPEL existentes en el repositorio no precisen de demasiados cambios para adaptarse al nuevo motor. Siendo así, los requisitos establecidos para este punto son los siguientes:

Soporte de WS-BPEL 2.0 Debe implementar casi todo o todo WS-BPEL 2.0. (Obligatorio).

Soporte de WSDL 1.1 Debe implementar por lo menos el subconjunto WS-I Basic Profile [1] 1.1 [19] de WSDL 1.1 [20]. (Obligatorio).

Ejemplos disponibles Debería ser capaz de ejecutar todos los ejemplos que tenemos sin cambios, o con cambios menores (para las composiciones que no cumplen el WS-I Basic Profile 1.1). (Obligatorio).

Otras tecnologías WS-* o BPEL Puede ser interesante ver todas las tecnologías WS-* y BPEL* que permite. Por ejemplo, WS-Security [21] o BPEL4People [22]. (Opcional).

3.2. Búsqueda y selección del motor WS-BPEL

Una vez establecidos todos los requisitos, el siguiente paso es buscar todos los motores WS-BPEL existentes hasta el momento.

Esta búsqueda de motores WS-BPEL se ha realizado a través de la web. Para ello se han utilizado una serie de herramientas útiles para la búsqueda de proyectos. La fuente de información más importante es el motor de búsqueda de contenido en internet Google.

El sitio web Ohloh ha sido otra fuente de información importante. Al recuperar datos de los repositorios de control de versiones (como CVS, SVN, o Git), Ohloh proporciona estadísticas acerca de la longevidad de los proyectos, sus licencias y las cifras de software, como líneas de código fuente y las estadísticas de las versiones.

También se ha empleado GitHub que es una forja para alojar proyectos usando el sistema de control de versiones Git. Para cada proyecto alojado en GitHub existe una wiki para el intercambio de información y la documentación del proyecto así como una serie de gráficos detallado del desarrollo la actividad de los desarrolladores del proyecto, cantidad de cambios realizados y la frecuencia del código.

Además se ha investigado en GoogleCode (sitio de Google para desarrolladores) o por redes sociales como Facebook o Twitter.

Las palabras claves principales que se han utilizado en estas webs han sido: *bpel engine*, *bpel tools*, *web services engine*, *deploying bpel process*.

Realizando estas búsquedas y descartando los motores que están obsoletos, los motores objeto del estudio se encuentran en la tabla 3.1.

A continuación se estudia cada motor por separado, basándonos en los requisitos previamente establecidos.

[1] WS-I Basic Profile es una especificación que consta de un conjunto de especificaciones de servicios web no propietario junto con aclaraciones, ajustes, interpretaciones y ampliaciones de las especificaciones que promueven la interoperabilidad, como SOAP y WSDL.

Motores	Páginas web
ActiveBPEL [23]	http://www.activebpel.org/
ActiveVOS [24]	http://www.activevos.com
Apache Agila [25]	http://incubator.apache.org/agila/
Apache ODE [26]	http://ode.apache.org/
Apache ServiceMix [27]	http://servicemix.apache.org/index.html
Apache Tuscany [28]	http://tuscany.apache.org/
BizTalk Server [29]	http://www.microsoft.com/biztalk/en/us/default.aspx
bpel-g [30]	http://code.google.com/p/bpel-g/
eInsight Business Process Manager [31]	http://www.binaryspectrum.com/service-oriented_architecture/eInsightBusinessProcessManager.html
JBoss Fuse [32]	http://www.redhat.com/products/jbossenterprisemiddleware/fuse/
GASwerk [33]	http://gaswerk.sourceforge.net/index.html
IBM alphaWorks BPWS4J [34]	http://www.alphaworks.ibm.com/tech/bpws4j
Intalio BPMS [35]	http://bpms.intalio.com
jBPM [36]	http://www.jboss.org/jbpm
Oracle BPEL Process Manager [37]	http://www.oracle.com/technetwork/middleware/bpel/overview/index.html
OW2 Orchestra [38]	http://orchestra.ow2.org/xwiki/bin/view/Main/WebHome
Riftsaw [39]	http://www.jboss.org/riftsaw
SAP Exchange Infrastructure [40]	http://sap-press.de/download/dateien/751/sap_press_exchange_infra_engl.pdf
Virtuoso Universal Server [41]	http://docs.openlinksw.com/virtuoso/
WSO2 Business Process Server [42]	http://wso2.com/products/business-process-server/

Tabla 3.1.: Motores WS-BPEL

Motores	Licencia	Reconocida por OSI	Compatible con ASL 2.0.	Desarrollo activo	Soporte
ActiveBPEL	GPL v2	Sí	No	No	No
ActiveVOS	Propietario	-	-	-	-
Apache Agila	Apache License v2.0	Sí	Sí	No	Sí
Apache ODE	Apache License v2.0	Sí	Sí	Sí	Sí
Apache ServiceMix	Apache License v2.0	Sí	Sí	Sí	Sí
Apache Tuscany	Apache License v2.0	Sí	Sí	Sí	Sí
BizTalk Server	Propietario	-	-	-	-
bpel-g	GPL v2	Sí	No	Sí	No
eInsight Business Process Manager	Propietario	-	-	-	-
Fuse ESB	Apache License v2.0	Sí	Sí	Sí	Sí
GASwerk	Apache License v2.0	Sí	Sí	Sí	Sí
IBM alphaWorks BPWS4J	Propietario	-	-	-	-
Intalio BPMS	MPL junto a opciones propietarias	Sí	Sí	Sí	Sí
jBPM	Apache License v2.0	Sí	Sí	Sí	Sí
Oracle BPEL Process Manager	Propietario	-	-	-	-
OW2 Orchestra	LGPL	Sí	Sí	Sí	Sí
Riftsaw	LGPL v2.1	Sí	Sí	Sí	Sí
SAP Exchange Infrastructure	Propietario	-	-	-	-
Virtuoso Universal Server	GPL v2 y propietario	Sí	No	Sí	Sí
WSO2 Business Process Server	Apache License v2.0	Sí	Sí	Sí	Sí

Tabla 3.2.: Motores WS-BPEL: aspectos generales I

3.2.1. Aspectos del proyecto en general

Como podemos ver en la tabla 3.2, descartaremos aquellos motores cuyas licencias no se ajustan a los requisitos, como los motores propietarios o aquellos cuyas licencias no sean compatibles con ASL.

Haremos una excepción con el motor bpel-g, aunque su licencia no nos conviene lo estudiaremos ya que está basado en ActiveBPEL (nuestro actual motor) y queremos observar las diferencias y mejoras al respecto.

Siguiendo estos criterios los motores que nos interesan se encuentra en la tabla 3.3.

Indagando en estos proyectos observamos que aquellos que han sido marcados con (*) son proyectos que en su interior incorporan el motor Apache ODE. Es por esta razón que en un principio los dejaremos al margen, de manera que estudiaremos el motor Apache ODE y si nos interesa estudiaremos los motores de los proyectos que lo integran (por si hay algún cambio en ellos o incluyen plugins que nos interese, etc).

En la tabla 3.4 podemos ver los motores que continúan en el estudio.

Inspeccionando con más detalle jBPM y observamos que la tecnología WS-BPEL sólo funciona de jBPM 1 a jBPM 3. jBPM 5.2 (la última versión) utiliza BPMN 2.0 de forma nativa, así que tendremos que descartarlo.

Motores	Licencia	Reconocida por OSI	Compatible con ASL 2.0.	Desarrollo activo	Soporte
Apache ODE	Apache License v2.0	Sí	Sí	Sí	Sí
Apache ServiceMix (*)	Apache License v2.0	Sí	Sí	Sí	Sí
Apache Tuscany (*)	Apache License v2.0	Sí	Sí	Sí	Sí
bpel-g	GPL v2	Sí	No	Sí	No
Fuse ESB (*)	Apache License v2.0	Sí	Sí	Sí	Sí
GASwerk (*)	Apache License v2.0	Sí	Sí	Sí	Sí
Intalio BPMS (*)	MPL junto a opciones propietarias	Sí	Sí	Sí	Sí
jBPM	Apache License v2.0	Sí	Sí	Sí	Sí
OW2 Orchestra	LGPL	Sí	Sí	Sí	Sí
Riftsaw (*)	LGPL v2.1	Sí	Sí	Sí	Sí
WSO2 Business Process Server (*)	Apache License v2.0	Sí	Sí	Sí	Sí

Tabla 3.3.: Motores WS-BPEL: aspectos generales II

Motores	Licencia	Reconocida por OSI	Compatible con ASL 2.0.	Desarrollo activo	Soporte
Apache ODE	Apache License v2.0	Sí	Sí	Sí	Sí
bpel-g	GPL v2	Sí	No	Sí	No
jBPM	Apache License v2.0	Sí	Sí	Sí	Sí
OW2 Orchestra	LGPL	Sí	Sí	Sí	Sí

Tabla 3.4.: Motores WS-BPEL: aspectos generales III

Motores	Licencia	Reconocida por OSI	Compatible con ASL 2.0.	Desarrollo activo	Soporte
Apache ODE	Apache License v2.0	Sí	Sí	Sí	Sí
bpel-g	GPL v2	Sí	No	Sí	No
OW2 Orchestra	LGPL	Sí	Sí	Sí	Sí

Tabla 3.5.: Motores WS-BPEL: aspectos generales IV

Motores	Lenguaje de Implementación	Compatibilidad con Linux y OpenJDK	Compilable	Integración con Maven
Apache ODE	Java, XML, JavaScript	Sí	Sí	Sí
bpel-g	Java, XML	Sí	Sí	Sí
OW2 Orchestra	Java, XML, HTML	Sí	Sí	Sí

Tabla 3.6.: Motores WS-BPEL: aspectos de implementación I

La cantidad de motores a estudiar se reducen a tres. Podemos verlo en la tabla 3.5.

Llegados a este momento, es hora de profundizar un poco más en las características de cada motor.

3.2.2. Aspectos de implementación

Como podemos observar en la tabla 3.6, los tres motores nos interesan a nivel de implementación.

Los tres compilan bien y se instalan correctamente, pero a la hora de iniciar el motor, Orchestra nos da una serie de problemas que intentamos solucionar mediante los foros y correos electrónicos oficiales, sin obtener respuesta alguna.

Por lo tanto, hemos de descartar Orchestra ya que no parece tener un seguimiento activo, no recibe soporte. Los motores que continúan en el estudio se encuentran en la tabla 3.7.

3.2.3. Aspectos de uso

En esta sección estudiaremos las ventajas e inconvenientes de los motores a la hora de descargarlos, instalarlos y ejecutarlos. Observamos sus características en la tabla 3.8.

Con respecto a su uso, ambos motores se instalan muy fácilmente y ambos presentan una interfaz web con la que podemos realizar un seguimiento de los procesos bpel,

Motores	Lenguaje de Implementación	Compatibilidad con Linux y OpenJDK	Compilable	Integración con Maven
Apache ODE	Java, XML, JavaScript	Sí	Sí	Sí
bpel-g	Java, XML	Sí	Sí	Sí

Tabla 3.7.: Motores WS-BPEL: aspectos de implementación II

Motores	Instalación sencilla	Interfaz web	Integración con IDE y BPELUnit	Documentación
Apache ODE	Sí	Sí	Sí	Sí
bpel-g	Sí	Sí	No	No

Tabla 3.8.: Motores WS-BPEL: aspectos de uso

desplegarlos, plegarlos, etc. Sin embargo encuentro una diferencia importante entre ellos, podemos encontrar documentación buena, tutoriales, blogs y foros activos de Apache ODE mientras que la documentación de bpel-g es escasa, no existen tutoriales y apenas hay blogs o foros en la web.

Llegados a este punto podemos concluir que bpel-g no nos interesa.

Finalmente nos queda Apache ODE.

3.2.4. Aspectos de tecnologías SOAP

Terminaremos el estudio comprobando las tecnologías SOAP de dicho motor (tabla 3.9).

El motor seleccionado es Apache ODE (figura 3.1).

Motores	Soporte WS-BPEL 2.0	Soporte de WSDL 1.1	Otras tecnologías
Apache ODE	Sí	Sí	SOAP 1.1 BPEL4WS 1.1 WS-I Basic Profile 1.1

Tabla 3.9.: Motores WS-BPEL: aspectos de tecnologías SOAP

Figura 3.1.: Logotipo del motor seleccionado.

3.3. Apache ODE

Apache ODE (*Orchestration Director Engine*) [26] es una herramienta desarrollada por Apache Software Fundation [43] bajo la licencia Apache, que permite la ejecución de procesos de negocio ejecutables basados en el estándar WS-BPEL 2.0. Sus características principales son:

1. Soporte del estándar WS-BPEL 2.0 de OASIS y el BPEL4WS 1.1.
2. Soporta dos capas de comunicación: una basada en Axis2 (Web Services http transport) y otro basado en el estándar JBI (usando ServiceMix).
3. Soporte para HTTP WSDL Binding.
4. Proporciona una API de alto nivel para acceso al motor que permite la integración del núcleo, con múltiples capas de comunicación e incluso su uso de manera independiente desde una aplicación.
5. La compilación de WS-BPEL provee de análisis detallados y validación desde la línea de comandos o durante el despliegue.
6. Interfaz para la gestión de procesos, instancias y mensajes.

La arquitectura de Apache ODE está dividida en diversos componentes, facilitando su reutilización. Los componentes clave de la arquitectura de ODE incluyen *ODE BPEL*

Compiler, *ODE BPEL Engine Runtime*, *ODE Data Access Objects (DAOs)*, *ODE Integration Layers (ILs)* y las herramientas de usuario. Una representación de alto nivel de las relaciones entre estos componentes se muestra en la figura 3.2.

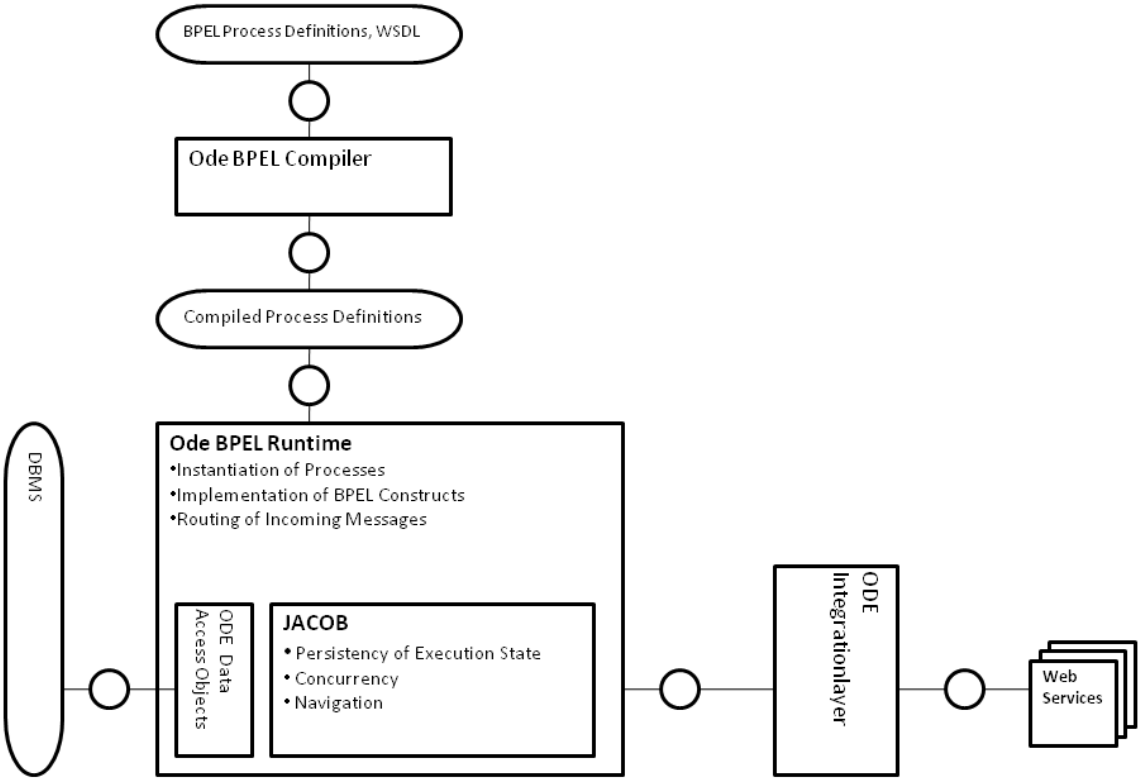

Figura 3.2.: Arquitectura del motor Apache ODE

ODE BPEL Compiler. Es el componente que compila el proceso WS-BPEL. Recibe el documento BPEL y sus dependencias (WSDL, Schemas, etc). La salida del compilador es o bien la compilación del proceso WS-BPEL compilado correctamente, o una lista de los mensajes de error que indica los problemas con los artefactos de origen.

ODE BPEL Engine Runtime ("tiempo de ejecución"). Este componente se encarga de ejecutar el proceso WS-BPEL compilado.

ODE Data Access Objects. Este componente hace de intermediario entre el componente ODE BPEL Engine Runtime (motor en tiempo de ejecución) y un almacén de datos subyacente (normalmente es una base de datos relacional JDBC).

ODE Integration Layers. Permite al componente ODE BPEL Engine Runtime (motor en tiempo de ejecución) comunicarse con el mundo exterior.

4. Adaptación de composiciones WS-BPEL de ActiveBPEL a Apache ODE

Una vez seleccionado el motor es el momento de adaptar las composiciones WS-BPEL del grupo. UCASE cuenta con un amplio repositorio de composiciones de servicios web escritos en WS-BPEL como *LoanApproval*, *LoanApprovalExtended*, *MetaSearch*, *SquaresSum*, *TravelReservationService*, *MarketPlace*, *TacService*, etc.

En este PFC se han seleccionado cuatro composiciones que modelan todos los casos que podemos encontrarnos al trabajar con las composiciones del grupo. Estas composiciones son las siguientes:

- LoanApprovalDoc: utiliza el estilo de codificación WSDL Document.
- LoanApprovalRPC: emplea el estilo de codificación WSDL RPC o llamadas a procedimientos remotos.
- MarketPlace: utiliza mensajes asíncronos.
- TacService: emplea hojas XSLT para la transformación de la salida.

A continuación describiremos detalladamente el comportamiento de cada una de estas composiciones y las analizaremos para conseguir que se ejecuten con el motor Apache ODE.

4.1. Estudio de las composiciones web

4.1.1. LoanApproval

LoanApproval es una composición de Gestión de Préstamos. Consiste en un proceso que recibe una solicitud de préstamo de un cliente y determina si la solicitud es aprobada o denegada.

El proceso se desarrolla de la siguiente forma:

Existe un límite en la cantidad de dinero del préstamo, en nuestro caso 10000 unidades monetarias, y según sea la cantidad de dinero solicitada por cliente se pueden dar los siguientes casos:

- El cliente solicita una cantidad menor o igual a 10000 unidades monetarias: En este caso interviene en la operación un asesor, que determinará si el riesgo de conceder este préstamo al cliente es alto (existe riesgo a la hora de conceder el préstamo) o bajo (existe muy poco riesgo a la hora de conceder el préstamo).
  - Si la respuesta del asesor es “low” (bajo) el préstamo será concedido directamente obteniendo por respuesta “true” (aceptado).
  - Si la respuesta del asesor es “high” (alto) será el aprobador el encargado de tomar la decisión final.
    - Si la respuesta del aprovador es “true” (aceptado) el préstamo será concedido y el cliente obtendrá la respuesta “true” (aceptado).
    - Si la respuesta del aprovador es “false” (denegado) el préstamo será rechazado y el cliente obtendrá la respuesta “false” (denegado).
- El cliente solicita una cantidad mayor a 10000 unidades monetarias: En este caso interviene directamente el aprobador, siendo él el responsable de conceder o denegar el préstamo.
  - Si la respuesta del aprovador es “true” (aceptado) el préstamo será concedido y el cliente obtendrá la respuesta “true” (aceptado).
  - Si la respuesta del aprovador es “false” (denegado) el préstamo será rechazado y el cliente obtendrá la respuesta “false” (denegado).

Para facilitar el estudio de esta composición, podemos ver su lógica en la figura 4.1.

En el grupo de investigación hay dos implementaciones de LoanApproval, una es de estilo RPC/literal es un estilo mas estricto con la especificación SOAP 1.1. y otra de estilo Document/literal Este estilo es un poco menos restrictivo, es decir, no tiene por qué seguir las convenciones específicas de SOAP. El mensaje SOAP se envía como un “documento” en el elemento <soap:Body>sin reglas de formato adicionales que tengan que ser consideradas.

4.1.2. MarketPlace

La composición MarketPlace o Mercado de compraventa consiste en una especie de puja, el vendedor lanza un producto al mercado con un precio de venta inicial y el comprador para poder adquirir el producto ofrece un precio mayor o igual al precio inicial establecido por el vendedor.

El proceso se desarrolla de la siguiente forma:

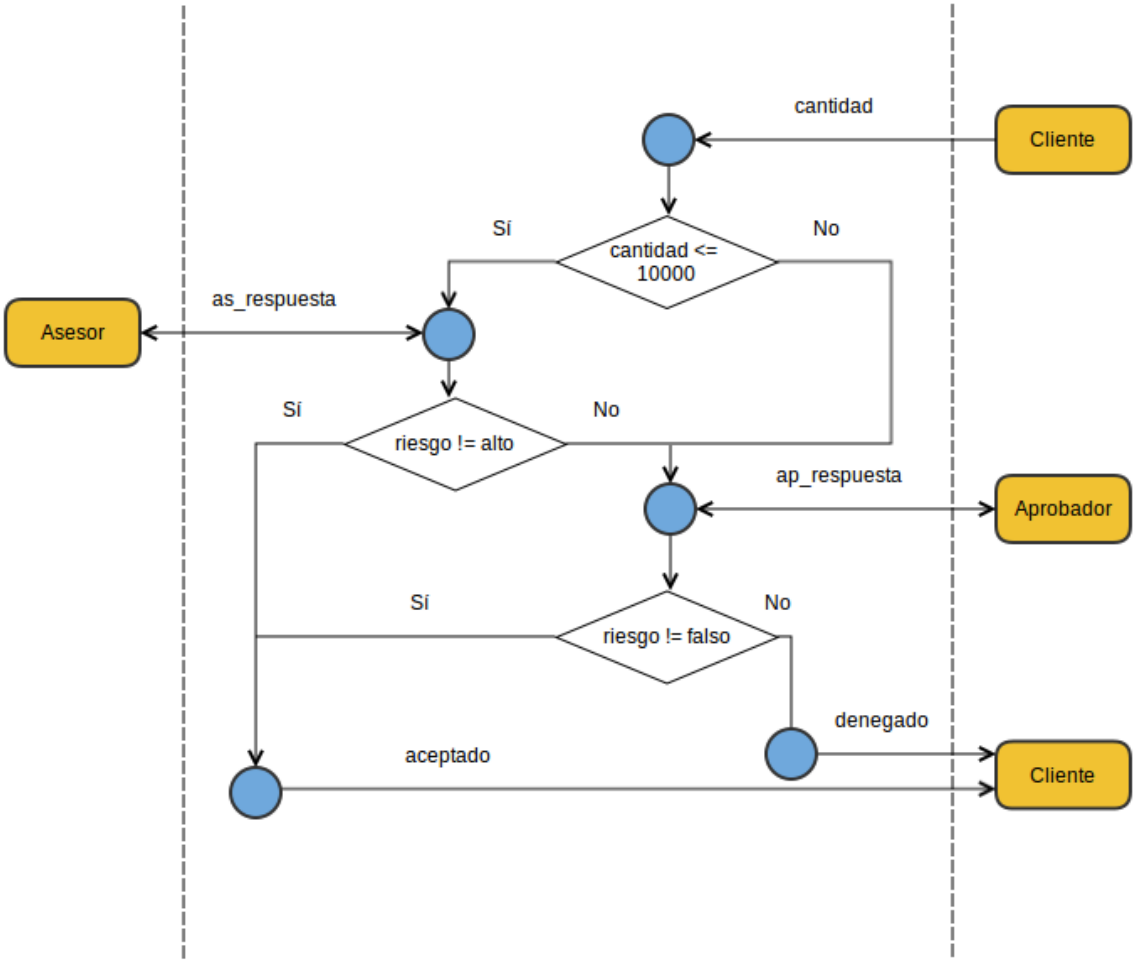

Figura 4.1.: Lógica de la composición LoanApproval

- El vendedor ofrece su producto y estipula un precio inicial de venta.
- El comprador interesado por este producto da un precio de compra para adquirirlo.
    - Si el precio de compra es igual o mayor al precio de venta, se efectuará la compra y ambos agentes obtendrán la respuesta “Deal Successful” (negocio satisfactorio).
    - Si el precio de compra es menor al precio de venta, no se efectuará la compra y por consiguiente ambos agentes obtendrán la respuesta “Deal Failed” (negocio fracasado).

Podemos observar su lógica visualmente en la figura 4.2.

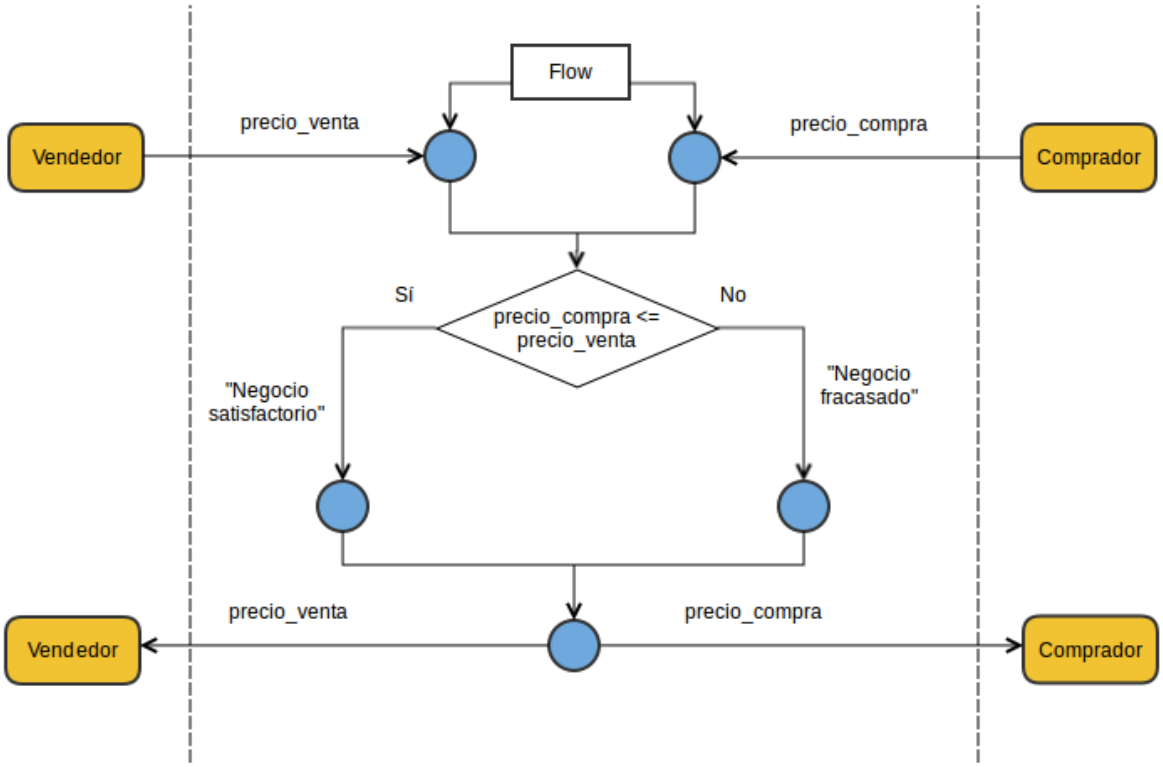

Figura 4.2.: Lógica de la composición MarketPlace

De esta composición cabe destacar el uso de la actividad `flow` de manera que se crea dos instancias, una para cada agente, que se ejecutan en paralelo.

4.1.3. TacService

La composición TacService invierte las líneas que se le envían. Para realizar esta inversión la composición se apoya en una hoja de transformaciones XSLT:

```
1  <stylesheet version="1.0" xmlns="http://www.w3.org/1999/XSL/Transform"
2  xmlns:p="http://xml.netbeans.org/schema/tacService">
3    <template match="p:tacRequest">
4      <p:tacResponse>
5         <apply-templates/>
6      </p:tacResponse>
7    </template>
8    <template match="p:lines">
9      <p:lines>
10       <for-each select="p:line">
11       <sort select="position()" data-type="number" order="descending"/>
12         <p:line><value-of select="." /></p:line>
13       </for-each>
14     </p:lines>
15   </template>
16 </stylesheet>
```

Podemos observar que la primera plantilla busca el nodo “p:tacRequest” y lo sustituye por un nodo respuesta `<p:TacResponse>`. Dentro de este nodo se llama a la siguiente plantilla que encuentra la línea “p:line” y la invierte imprimiendo el resultado en el elemento de respuesta.

Podemos observar su lógica en la imagen 4.3.

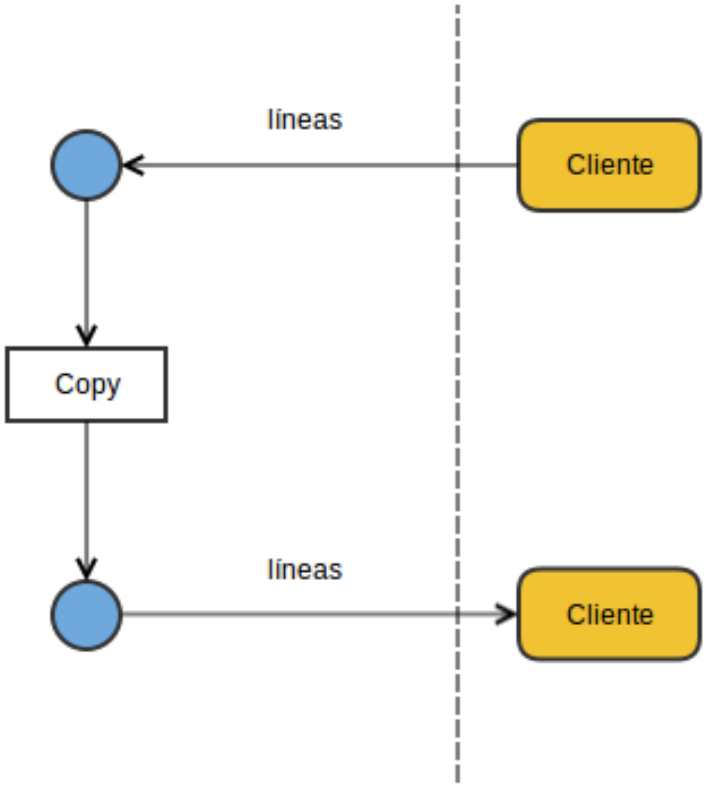

Figura 4.3.: Lógica de la composición TacService

4.2. Desarrollo del protocolo de adaptación de composiciones WS-BPEL

El protocolo de adaptación de composiciones WS-BPEL dirigidas de ActiveBPEL a Apache ODE es una serie de instrucciones que establecen cómo se debe actuar para adaptar una composición WS-BPEL desarrollada para ser ejecutada con el motor ActiveBPEL a ser ejecutada con el motor Apache ODE.

En primer lugar, hemos adaptado las cuatro composiciones seleccionadas. De los errores y problemas encontrados en el proceso obtenemos las instrucciones que componen este protocolo. Veamos el proceso.

4.2.1. Errores especificados en el estándar WS-I Basic Profile

Apache ODE es muy estricto con el estándar WS-I Basic Profile por lo que en la mayoría de los casos en los que obtengamos algún error estará definido en este estándar.

A continuación listamos los errores que nos hemos encontrado al adaptar las composiciones indicando en que sección del estándar WS-I Basic Profile se encuentran definidos:

- LoanApprovalDoc
  - Sigue el estándar.
- LoanApprovalRPC
  - Sección 4.7.10 - Namespaces for soapbind Elements.
  - Sección 4.7.19 - Response Wrappers.
  - Sección 4.7.20 - Part Accessors.
- MarketPlace
  - Sección 4.7.10 - Namespaces for soapbind Elements.
  - Sección 4.5.1 - Ordering of part Elements.
  - Cambios en namespaces.
- TacService
  - Sigue el estándar.

Veamos estas secciones:

- Sección 4.7.10 - Namespaces for soapbind Elements:
  - Estilo document-literal: Los elementos `soap:body`, `soap:header`, `soap:headerfault` y `soap:fault` NO DEBEN tener el atributo `namespace` especificado.
  - Estilo RPC-literal: El elemento `soap:body` DEBE contener el atributo `namespace` especificado (con la URI absoluta). Los elementos `soap:header`, `soap:headerfault` y `soap:fault` NO DEBEN tener el atributo `namespace` especificado.

Ejemplo de definición de un binding con ActiveBPEL:

```
...

<binding name="ApprovalBinding" type="ns0:loanApprovalPT">
   <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
   <operation name="approve">
      <soap:operation/>
      <input name="input1">
         <soap:body use="literal"/>
      </input>
      <output name="output1">
         <soap:body use="literal"/>
      </output>
      <fault name="fault1">
         <soap:fault name="fault1" use="literal"/>
      </fault>
   </operation>
</binding>

...
```

Transformación del binding con Apache ODE:

```
...

<binding name="ApprovalBinding" type="ns0:loanApprovalPT">
   <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
   <operation name="approve">
      <soap:operation/>
      <input name="input1">
         <soap:body use="literal" namespace="http://services"/>
      </input>
      <output name="output1">
         <soap:body use="literal" namespace="http://services"/>
      </output>
      <fault name="loanProcessFault">
         <soap:fault name="loanProcessFault" use="literal"/>
      </fault>
   </operation>
</binding>

...
```

En realidad esto sería un error de BPELUnit. Nosotros nos apoyamos en el uso de BPELUnit para ejecutar las pruebas de la composición y es BPELUnit el que debería de añadirle este espacio de nombre. Mientras que no se solucione el problema con BPELUnit los espacios de nombre se añadirán manualmente.

- Sección 4.7.19 - Response Wrappers:
  - Estilo rpc-literal: El mensaje de respuesta debe tener un elemento de envoltura cuyo nombre se corresponde con el nombre del elemento `wsdl:operation` más el sufijo “Response”.

Este problema fue mayor debido a que BPELUnit no devolvía los mensajes de respuesta en el formato correcto, faltaba añadirle el sufijo “Response” por lo tanto los mensajes devueltos eran totalmente incompatibles con el motor Apache ODE. Para solucionarlo fue necesario contactar con uno de los desarrolladores de BPELUnit, le informamos

del error y finalmente mejoraron el programa para que la salida de los mensajes de BPELUnit con composiciones de estilo rpc-literal fuese compatible con Apache ODE.

- Sección 4.7.20 - Part Accessors:
  - Estilo rpc-literal: Un mensaje descrito en un binding de tipo rpc-literal DEBE tener el elemento part accessor para los parámetros y devuelve un valor sin namespace. Los elementos part accessor en un mensaje de este tipo DEBEN tener un nombre local del mismo valor que el atributo `name` del correspondiente elemento `wsdl:part`.
- Sección 4.5.1 - Ordering of part Elements:
  - El orden de los elementos en el cuerpo(`soap:body`) de un mensaje debe ser el mismo que en las partes (`wsdl:parts`) en el mensaje (`wsdl:message`) que lo describe.

Dados los siguientes mensajes definidos en el fichero marketplace.wsdl:

```
1  ...
2
3  <wsdl:message name="sellerInfoMessage">
4     <wsdl:part name="inventoryItem" type="xsd:string"/>
5     <wsdl:part name="askingPrice" type="xsd:integer"/>
6  </wsdl:message>
7
8  <wsdl:message name="buyerInfoMessage">
9     <wsdl:part name="item" type="xsd:string"/>
10    <wsdl:part name="offer" type="xsd:integer"/>
11 </wsdl:message>
12
13 <wsdl:message name="negotiationMessage">
14    <wsdl:part name="outcome" type="xsd:string"/>
15 </wsdl:message>
16
17 ...
```

En el siguiente ejemplo se define un caso de prueba para la composición MarketPlace de ActiveBPEL y el mismo caso de prueba para Apache ODE. En el caso de prueba existe un comprador que ofrece la cantidad de 150 unidades monetarias (línea 36) para adquirir el producto en venta (línea 35). El vendedor establece el precio mínimo en 100 unidades monetarias (línea 13) por lo que la compra se efectúa correctamente.

ActiveBPEL es capaz de soportar un caso de prueba donde el orden de las partes de los mensajes no se escriben correctamente (a pesar de ello este motor los ejecuta en el orden correcto). Veamos el ejemplo:

```
1  <tes:testCases>
2
3     <tes:testCase name="SuccessfulDeal" basedOn="" abstract="false" vary="true">
4      <tes:clientTrack>
5        <tes:sendReceive
6           service="tns:marketplaceSeller"
7           port="seller"
8           operation="submit">
9
```

```
10        <tes:send fault="false" delaySequence="0,2">
11          <tes:template>
12            <tns:inventoryItem>Takuan-$testCaseName</tns:inventoryItem>
13            <tns:askingPrice>100</tns:askingPrice>
14          </tes:template>
15        </tes:send>
16
17        <tes:receive fault="false">
18          <tes:condition>
19            <tes:expression>outcome</tes:expression>
20            <tes:value>'Deal Successful'</tes:value>
21          </tes:condition>
22        </tes:receive>
23
24      </tes:sendReceive>
25    </tes:clientTrack>
26
27    <tes:partnerTrack name="Buyer">
28      <tes:sendReceive
29         service="tns:marketplaceBuyer"
30         port="buyer"
31         operation="submit">
32
33        <tes:send fault="false" delaySequence="2,0">
34          <tes:template>
35            <tns:inventoryItem>Takuan-$testCaseName</tns:inventoryItem>
36            <tns:askingPrice>150</tns:askingPrice>
37          </tes:template>
38        </tes:send>
39
40        <tes:receive fault="false"
41          <tes:condition>
42            <tes:expression>outcome</tes:expression>
43            <tes:value>'Deal Successful'</tes:value>
44          </tes:condition>
45        </tes:receive>
46
47      </tes:sendReceive>
48    </tes:partnerTrack>
49
50  </tes:testCase>
51  ...
```

En la línea 6 opera el servicio *marketplaceSeller* que esta relacionado con el mensaje *sellerInfoMessage* y si observamos las partes que se describen en las líneas 12 y 13 son correctas. Pero en el siguiente servicio que se encuentra en la línea 29 cuyo nombre es *marketplaceBuyer* se relaciona con el mensaje *buyerInfoMessage* y sin embargo las partes son incorrectas (no son las partes correspondientes al mensaje *buyerInfoMessage*). ActiveBPEL es más flexible en este aspecto ya que a pesar de no haber escrito bien el orden de las partes de los mensajes, los interpreta correctamente.

Mientras que Apache ODE necesita establecer el orden y los nombres exactos. Veamos la misma definición del caso de prueba adaptada para Apache ODE:

```
1  <tes:testCases>
2
3    <tes:testCase name="SuccessfulDeal" basedOn="" abstract="false" vary="true">
4      <tes:clientTrack>
5        <tes:sendReceive
6          service="tns:marketplaceSeller"
7          port="seller"
```

```
8          operation="submit">
9
10        <tes:send fault="false" delaySequence="0,2">
11          <tes:template>
12            <inventoryItem>Takuan-$testCaseName</inventoryItem>
13            <askingPrice>100</askingPrice>
14          </tes:template>
15        </tes:send>
16
17        <tes:receive fault="false">
18          <tes:condition>
19            <tes:expression>outcome</tes:expression>
20            <tes:value>'Deal Successful'</tes:value>
21          </tes:condition>
22        </tes:receive>
23
24      </tes:sendReceive>
25    </tes:clientTrack>
26
27    <tes:partnerTrack name="Buyer">
28      <tes:sendReceive
29         service="tns:marketplaceBuyer"
30         port="buyer"
31         operation="submit">
32
33        <tes:send fault="false" delaySequence="2,0">
34          <tes:template>
35            <item>Takuan-$testCaseName</item>
36            <offer>150</offer>
37          </tes:template>
38        </tes:send>
39
40        <tes:receive fault="false">
41          <tes:condition>
42            <tes:expression>outcome</tes:expression>
43            <tes:value>'Deal Successful'</tes:value>
44          </tes:condition>
45        </tes:receive>
46
47      </tes:sendReceive>
48    </tes:partnerTrack>
49
50  </tes:testCase>
51
52 ...
```

Como podemos observar en las líneas 35 y 36 se han utilizado las partes correspondientes a este servicio.

También podemos obtener otro tipo de errores como:

- Los espacios de nombre cambian. En alguna ocasión se ha dado que algún espacio de nombre establecido en una versión antigua de una composición ha cambiado y por lo tanto los elementos definidos con él no están bien calificados.

Definición antigua de bpws:

```
1 <definitions targetNamespace= ...
2          xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
3
4 ...
```

Definición nueva de bpws:

```
<definitions targetNamespace= ...
         xmlns:bpws="http://docs.oasis-open.org/wsbpel/2.0/varprop"

...
```

4.2.2. Diferencias en la dirección de la composición

En los ficheros WSDL se establecen las direcciones de los servicios con los que trabajan los procesos, y en concreto se establece la dirección de despliegue de la composición. Esta dirección es diferente según el motor de ejecución que se utilice por lo que tendremos que retocarla para que funcione con Apache ODE.

```
http://localhost:8080/active-bpel/services/ ...
```

```
http://localhost:8080/ode/processes/ ...
```

4.2.3. Diferencias en la definición de los casos de prueba

BPELUnit necesita obtener información sobre la forma en la que tiene que desplegar el proceso que va a poner a prueba. Es por ello que en los ficheros BPTS se encuentra un elemento denominado `deployment` en el que le indicamos el motor en el que se va a desplegar y las propiedades necesarias para llevarlo a cabo. Estas propiedades son diferentes según el motor que se utilice de manera que hay que retocar estas líneas.

En ActiveBPEL viene definido así:

```
...

<tes:deployment>
   <tes:put name="marketplace" type="activebpel">
      <tes:wsdl>marketplace.wsdl</tes:wsdl>
      <tes:property name="BPRFile">marketplace.bpr</tes:property>
   </tes:put>
   <tes:partner name="Buyer" wsdl="marketplace.wsdl"/>
</tes:deployment>

...
```

Modificado para Apache ODE sería:

```
...

<tes:deployment>
   <tes:put name="marketplace" type="ode">
      <tes:wsdl>marketplace.wsdl</tes:wsdl>
      <tes:property name="ODEDeploymentServiceURL">http://localhost:8080/ode/processes/
          DeploymentService</tes:property>
      <tes:property name="DeploymentArchive">/home/olga/Documentos/adaptacionesODE/workspace/
          MarketPlace</tes:property>
   </tes:put>
```

```
9     <tes:partner name="Buyer" wsdl="marketplace.wsdl"/>
10 </tes:deployment>
11
12 ...
```

4.2.4. Generar el descriptor de despliegue de Apache ODE

El archivo deploy.xml es un descriptor de despliegue XML que se utiliza para configurar una unidad de despliegue de Apache ODE.

Generar este fichero no es un trabajo sencillo, tenemos que estudiar los ficheros WS-BPEL y WSDL para obtener los datos necesarios para generarlo.

Por este motivo, para facilitar la adaptación de las composiciones web, se ha creado una función que lee un archivo BPEL y genera automáticamente el fichero descriptor de despliegue asociado a esa composición.

Un ejemplo simple de un descriptor de despliegue es el siguiente:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <deploy xmlns="http://www.apache.org/ode/schemas/dd/2007/03"
3        xmlns:ns2="http://j2ee.netbeans.org/wsdl/ApprovalService"
4        xmlns:ns1="http://j2ee.netbeans.org/wsdl/ConcreteAssessorService"
5        xmlns:bpel="http://enterprise.netbeans.org/bpel/LoanApproval_V2/loanApprovalProcess"
6        xmlns:ns3="http://j2ee.netbeans.org/wsdl/loanServicePT"
7        xmlns:ns0="http://j2ee.netbeans.org/wsdl/ConcreteLoanService">
8     <process name="bpel:loanApprovalProcess">
9        <active>true</active>
10       <retired>false</retired>
11       <process-events generate="all"/>
12       <invoke partnerLink="approver">
13          <service name="ns2:ApprovalService" port="ApprovalServicePort"/>
14       </invoke>
15       <invoke partnerLink="assessor">
16          <service name="ns1:RiskAssessmentService" port="RiskAssessmentPort"/>
17       </invoke>
18       <provide partnerLink="customer">
19          <service name="ns0:LoanService" port="LoanServicePort"/>
20       </provide>
21    </process>
22 </deploy>
```

Veamos las características de este tipo de archivo:

- Se compone de un elemento raíz denominado `deploy` que contiene una lista de todos los procesos desplegados desde el directorio de implementación.
- Cada proceso se identificará por su nombre completo mediante el atributo `name`.
- Por defecto las características del proceso serán las siguientes:
    - Establecemos el proceso en activo mediante el elemento `active` con valor “true”.

- El proceso no estará retirado. Lo indicamos mediante el elemento `retired` con valor “false”.
- El nivel de monitorización de los procesos será completo. Lo habilitamos mediante el elemento `process-events` con valor “all”.

- Cada proceso debe enumerar los servicios invocados y prestados por el proceso mediante los elementos `invoke` y `provide` respectivamente.
- Cada elemento `invoke` y `provide` debe especificar el atributo `partnerLink` para poder establecer un enlace con el servicio.

El usuario que desee generar automáticamente este archivo utilizando la biblioteca ODE-integration realizará lo siguiente:

```
1 ODEDeploymentArchive odeDeploy;
2
3 BPELProcessDefinition bpel = new BPELProcessDefinition(new File("loanApprovalProcess.bpel"));
4 odeDeploy = new ODEDeploymentArchive(bpel);
5
6 XMLDocument deployXML = odeDeploy.generateDeploymentDescriptor();
7 deployXML.dumpToFile(new File(deploy.xml"));
```

1. Creamos una instancia de la clase *BPELProcessDefinition* con el archivo BPEL de nuestra composición.
2. Generamos una instancia de la clase *ODEDeploymentArchive*. Esta clase requiere el objeto de tipo *BPELProcessDefinition* que creamos en el paso anterior.
3. Llamamos a la función *generateDeploymentDescriptor()* de la clase *ODEDeploymentArchive* que nos devuelve un objeto *XMLDocument* con el texto del documento descriptor de despliegue de Apache ODE.
4. Por último generamos el fichero físico denominado deploy.xml en la carpeta propia de la composición.

4.3. Instrucciones del protocolo

Finalmente las instrucciones del protocolo se pueden resumir en los siguientes pasos:

- Comprobar que la composición cumple con el estándar WS-I Basic Profile.
- Retocar la dirección de la composición WS-BPEL definida en los servicios descritos en los documentos WSDL.
- Retocar la definición de los casos de prueba.

- Generar el fichero descriptor de despliegue. Esta función se encuentra automatizada en la biblioteca ODE-integration y en las herramientas en las que se integre por lo tanto aquellos que utilicen estas herramientas pueden ignorar esta última instrucción.

5. Desarrollo de la biblioteca ODE-integration

Una vez adaptadas las composiciones WS-BPEL es el momento de integrar el motor Apache ODE en las herramientas del grupo. Para facilitar esta integración se desarrolla una biblioteca mediante la que se automatiza todo el proceso de descarga e instalación del motor. Además se envuelve el motor en una clase para facilitar su manejo.

En primer lugar se realizará la primera versión de la biblioteca ODE-integration, que contará con las funcionalidades básicas para la ejecución de las composiciones WS-BPEL de manera que será capaz de desplegar una composición y ejecutar las pruebas unitarias con BPELUnit y lo más importante, esta versión ya estará preparada para ser integrada en la herramienta MuBPEL.

Una vez que se inicia el proceso de integración en MuBPEL se realizaran sucesivas modificaciones de manera que en cada una de ellas se añade una funcionalidad y por lo tanto se mejora la calidad de la biblioteca hasta obtener la versión final.

A continuación explicaremos con más detalle el desarrollo de esta biblioteca y analizaremos su funcionalidad mediante un estudio de ingeniería.

5.1. Análisis

En esta sección analizamos la biblioteca ODE-integration, especificamos los requisitos que debe cumplir y la funcionalidad de la misma mediante el modelo de casos de uso, el modelo conceptual de datos y los diagramas de secuencia del sistema.

5.1.1. Requisitos Funcionales

Los requisitos funcionales definen las funciones que el sistema será capaz de realizar. Estas funciones son:

- Descarga e instalación automática de Apache ODE. Un requisito necesario a cumplir por esta biblioteca es que la descarga e instalación del motor sea transparente para el usuario ya que así debe de ser en las herramientas en las que se integre Apache ODE.

- Despliegue de composiciones WS-BPEL. El sistema debe preparar los ficheros necesarios para que las composiciones WS-BPEL sean desplegadas con el motor Apache ODE.
- Ejecución de una composición WS-BPEL frente a su conjunto de casos de prueba. Como resultado, se debe generar un fichero con los resultados de la ejecución, que indicarán si los casos de prueba se han pasado con éxito o no.

5.1.2. Requisitos de Implementación

Los requisitos de implementación son:

- El lenguaje de implementación será Java. Es necesario desarrollar la biblioteca en este lenguaje ya que las herramientas en las que se va a integrar están desarrolladas en Java.
- Realización de pruebas unitarias. Es necesario realizar las suficientes pruebas unitarias para asegurarnos del correcto funcionamiento del código.
- Utilizar un entorno de integración continua. De esta manera podemos asegurarnos en todo momento que la biblioteca sigue cumpliendo correctamente con sus funcionalidades.

5.1.3. Automatización del motor Apache ODE

En el desarrollo de ODE-integration cabe destacar la forma en la que se ha automatizado todo el proceso de descarga del motor. Se ha utilizado el motor Apache ODE 1.3.5 desplegado como un WAR en un servidor de aplicaciones Tomcat 7.0. Veamos este proceso.

5.1.3.1. Integración del servidor de aplicaciones J2EE Tomcat

Apache Tomcat [44] es un servidor web con soporte de Java Servlets[1] y Java Servlet Pages (JSPs). Es un software de código abierto publicado bajo la licencia Apache Software License 2.0.

Su utilidad dentro de ODE-integration es la siguiente:

[1] Los servlets son objetos que se ejecutan dentro y fuera del contexto de un contenedor de servlets (ej: Tomcat). El servlet es una clase en el lenguaje de programación Java, utilizada para ampliar las capacidades de un servidor. Aunque los servlets pueden responder a cualquier tipo de solicitudes, se utilizan comúnmente para extender las aplicaciones alojadas por servidores web, por lo que se pueden considerar como los applets de Java que se ejecutan en servidores en lugar de en los navegadores web.

- Desplegar los WS utilizados en el estudio.
- Desplegar y ejecutar Apache ODE como un servicio web en Axis 2 [45], mediante la implementación de la distribución WAR de ODE (ode.war).

Utilizaremos la última versión estable de Tomcat en el momento (versión 7.0).

Para entender mejor el funcionamiento de Tomcat estudiaremos la jerarquía de directorios:

- bin: arranque, cierre, y otros scripts y ejecutables.
- common: clases comunes que pueden utilizar Catalina y las aplicaciones web.
- conf: ficheros XML y los correspondientes DTD para la configuración de Tomcat.
- logs: logs de Catalina y de las aplicaciones.
- server: clases utilizadas solamente por Catalina.
- shared: clases compartidas por todas las aplicaciones web.
- webapps: directorio que contiene las aplicaciones web.
- work: almacenamiento temporal de ficheros y directorios.

La utilización del servidor Tomcat debe ser transparente al usuario, por este motivo lo hemos embebido en la biblioteca.

Para ello importamos la clase Tomcat (org.apache.catalina.startup.Tomcat) en la que encontraremos todo lo necesario para empotrar el servidor en nuestra biblioteca.

Debemos llamar a los siguientes métodos de esta clase en el orden establecido para asegurar un funcionamiento correcto.

- Crear una instancia nueva de esta clase.

```
1 private Tomcat tomcat;
2 tomcat  = new Tomcat();
```

- Establecer las propiedades relevantes del objeto creado.

```
1 tomcat.setPort(tomcatPort);
2 tomcat.setBaseDir(baseDir);
3 tomcat.setHostname(hostName);
4 configureLogging();
```

- Añadir y configurar un contexto webapp. Es en este paso donde tenemos que conectar Apache ODE con el servidor Tomcat. Para ello tenemos que copiar a la carpeta webapps de Tomcat el archivo WAR que se encuentra en el directorio raíz de la distribución WAR de ODE, renombraremos este archivo a “ode.war” y establecemos el contexto.

```
this.odeFile = pathToODE;
final File dirWebapps = new File(baseDir, "webapps");
FileUtils.copyFileToDirectory(odeFile, dirWebapps);

final File oldName = new File(dirWebapps, odeFile.getName());
final File newName = new File(dirWebapps, "ode.war");
oldName.renameTo(newName);

tomcat.addWebapp("/ode", "ode");
```

- Llamar a start() para iniciar Tomcat y Apache ODE estará en funcionamiento. Podemos comprobar que Apache ODE está ejecutandose viendo la página de bienvenida de Axis2 en la URL `http://localhost:8080/ode`.

```
tomcat.start();
```

- Llamar a stop() para parar el servidor y destroy() para destruirlo.

```
if (tomcat.getServer() != null &&
  tomcat.getServer().getState() != LifecycleState.DESTROYED) {
  if (tomcat.getServer().getState() != LifecycleState.STOPPED) {
    tomcat.stop();
  }
  tomcat.destroy();
}
```

5.1.3.2. Creación de la biblioteca ODE-integration: Maven

Maven se ha utilizado para la construcción de esta biblioteca. Podemos obtener más información sobre esta herramienta en la sección 5.3.1.3.

En esta sección detallaremos las dependencias necesarias que hay que añadir en Maven para utilizar Tomcat embebido y para poder descargar Apache ODE automáticamente.

- **Dependencias Tomcat 7**

```
<properties>
     <tomcat.version>7.0.26</tomcat.version>
</properties>

<dependencies>
  <dependency>
     <groupId>org.apache.tomcat.embed</groupId>
     <artifactId>tomcat-embed-core</artifactId>
     <version>${tomcat.version}</version>
  </dependency>
  <dependency>
     <groupId>org.apache.tomcat.embed</groupId>
     <artifactId>tomcat-embed-logging-log4j</artifactId>
     <version>${tomcat.version}</version>
  </dependency>
  <dependency>
     <groupId>org.apache.tomcat.embed</groupId>
     <artifactId>tomcat-embed-jasper</artifactId>
     <version>${tomcat.version}</version>
  </dependency>
```

```
21
22   ...
23
24 </dependencies>
```

La primera dependencia (`tomcat-embed-core`) es el propio Tomcat "incrustable". La segunda dependencia (`tomcat-embed-logging-log4j`) es la aplicación de registro log4j para Tomcat embebido. La tercera dependencia (`tomcat-embed-jasper`) sirve para poder usar JSP, esta dependencia es necesaria para que funcione la web de administración.

- **Descarga de Apache ODE**

  Para poder descargar automáticamente el motor tenemos que añadirlo como una dependencia de Maven, así será éste el encargado de descargarse la distribución WAR de Apache ODE.

```
1  <dependencies>
2
3     ...
4
5     <dependency>
6        <groupId>org.apache.ode</groupId>
7        <artifactId>ode-axis2-war</artifactId>
8        <version>1.3.5</version>
9        <type>war</type>
10       <scope>runtime</scope>
11    </dependency>
12 </dependencies>
```

Como nos interesa conservar el archivo .war que está en la raíz de la distribución descargada, mediante Maven podemos copiar este fichero a una carpeta de nuestro proyecto. Utilizaremos el plugin de Maven: `maven-dependency-plugin`.

```
1  <build>
2     <plugins>
3        <plugin>
4           <artifactId>maven-dependency-plugin</artifactId>
5           <executions>
6              <execution>
7                 <id>copy-wars</id>
8                 <phase>process-resources</phase>
9                 <goals><goal>copy-dependencies</goal></goals>
10                <configuration>
11                   <includeTypes>war</includeTypes>
12                </configuration>
13             </execution>
14          </executions>
15       </plugin>
16
17       ...
18
19    </plugins>
20 </build>
```

De esta manera ya tenemos disponible el fichero ode-axis2-war-1.3.5.war en la carpeta target/dependency de nuestro proyecto.

5.1.4. Modelo de casos de uso

En esta sección describiremos las diferentes funcionalidades de la biblioteca ODE-integration mediante un diagrama de clases y la correspondiente descripción de los casos de uso que actúan.

De aquí en adelante, cuando hablemos de un usuario de la biblioteca nos referimos a otros sistemas externos que requieren utilizar alguna funcionalidad de la misma.

En la figura 5.1 podemos ver el diagrama de casos de uso de ODE-integration.

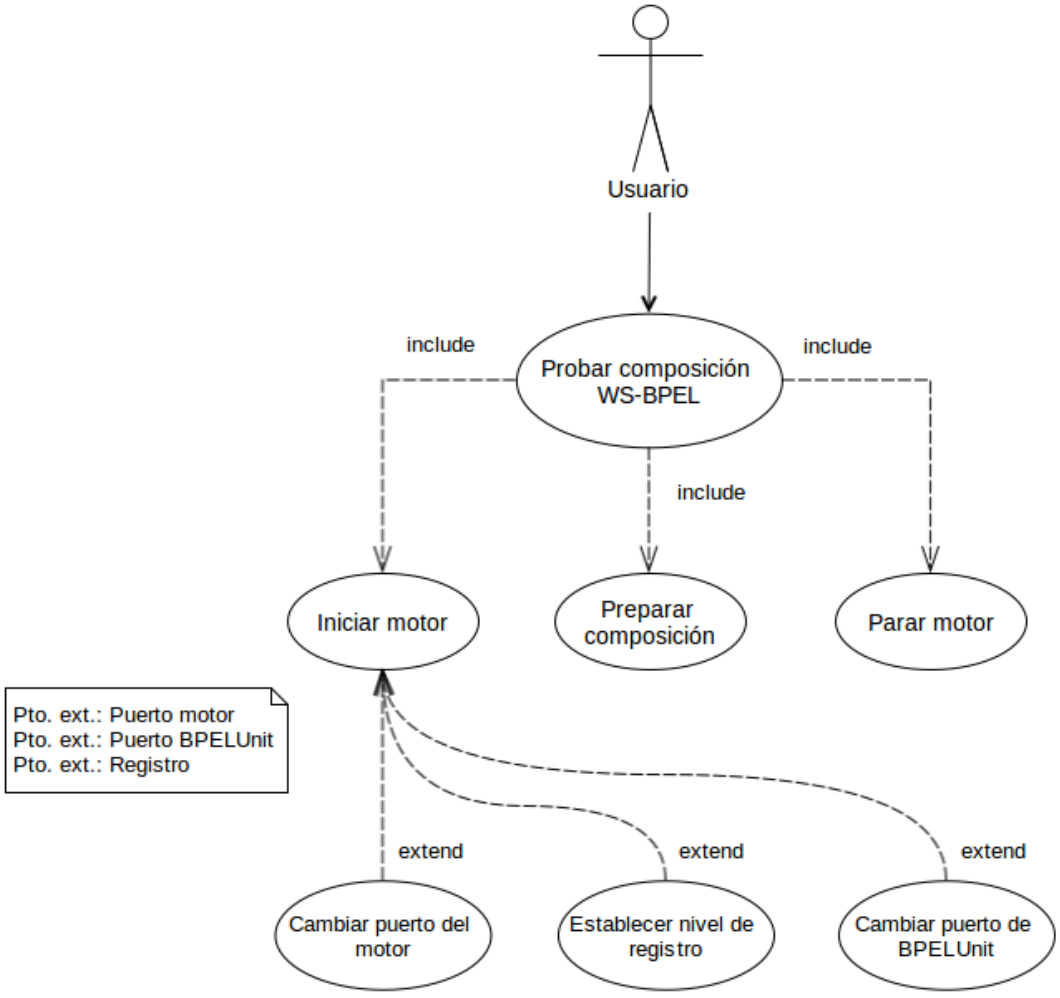

Figura 5.1.: Diagrama de casos de uso

5.1.4.1. Caso de uso: Probar composición WS-BPEL

Actor principal: Usuario
Personal involucrado e intereses:

- Usuario: Requiere ejecutar pruebas unitarias a una composición WS-BPEL.
- Sistema: Permite al usuario ejecutar las pruebas unitarias.

Precondiciones: Ninguna.
Postcondiciones: El sistema indica los datos necesarios a BPELUnit para que se ejecuten las pruebas de la composición y se devuelve un fichero con los resultados obtenidos por BPELUnit.
Escenario principal:

1. Include Iniciar motor.
2. Include Preparar composición.
3. El sistema informa a BPELUnit de la localización del fichero de pruebas BPTS.
4. El sistema informa a BPELUnit del puerto de ejecución del motor Apache ODE.
5. El sistema informa a BPELUnit del puerto de ejecución del mismo.
6. El sistema informa a BPELUnit de la localización de la composición adaptada (archivo zip generado en el paso 4).
7. El sistema informa a BPELUnit del nivel de registro que debe establecer.
8. El nivel de registro es “NONE”.
9. BPELUnit ejecuta las pruebas especificadas en el fichero BPTS.
10. BPELUnit devuelve los resultados de las pruebas en un fichero XML (/bpelunit[1-9]*.xml/) en la carpeta base de ejecución.
11. El sistema finaliza la ejecución del programa.

Escenarios de excepción:

2a. El fichero BPEL o el fichero BPTS no existen.

    1. El sistema informa del error y finaliza la ejecución del programa.

10a. El nivel de registro es distinto de “NONE”.

    1. BPELUnit crea una carpeta llamada “proccess-log” en la carpeta base de ejecución.
    2. BPELUnit ejecuta los casos de prueba y por cada uno de ellos genera un fichero de registro capturando los mensajes del nivel dado.
    3. Sigue en el paso 9.

*a. El usuario aborta la ejecución del caso de uso.

1. El sistema finaliza la ejecución del programa.

5.1.4.2. Caso de uso: Iniciar motor

Puntos de extensión:

- Puerto motor, paso 1.
- Registro, paso 2.
- Puerto BPELUnit, paso 3.

Actor principal: Sistema.
Personal involucrado e intereses:

- Sistema: Realiza las modificaciones oportunas para poder iniciar el servidor correctamente.

Precondiciones: Ninguna.
Postcondiciones: El sistema inicia el servidor Tomcat y ejecuta el motor Apache ODE.
Escenario principal:

1. El sistema comprueba que el puerto estándar (8080) no está en uso y establece este valor.
2. El sistema establece el nivel de registro a “NONE”.
3. El sistema establece el puerto de BPELUnit estándar (7777).
4. El sistema establece una carpeta temporal (/temp[1-9]+/) como carpeta base de ejecución.
5. El sistema crea un archivo de registro “tomcat.log” en la carpeta base con el nivel de ejecución “WARN” que recogerá el registro de ejecución del servidor Tomcat.
6. Se inicia el servidor Tomcat y con él Apache ODE.

Escenarios de excepción:

1a. El puerto 8080 está en uso.

1. El sistema informa del error y finaliza la ejecución del programa.

*a. El usuario aborta la ejecución del caso de uso.

1. El sistema finaliza la ejecución del programa.

5.1.4.3. Caso de uso: Cambiar puerto del motor

Activa: El usuario indica un puerto específico de ejecución del motor.
Punto de extensión: Puerto motor en caso de uso: Iniciar servidor.
Nivel: Función.
Actor principal: Usuario.
Personal involucrado e intereses:

- Usuario: Requiere cambiar el puerto de ejecución de Apache ODE.
- Sistema: Permite cambiar el puerto.

Precondiciones: Ninguna.
Postcondiciones: El sistema cambia el valor del puerto en el que se ejecuta el motor.
Escenario principal:

1. El usuario indica el valor del nuevo puerto de ejecución.
2. El sistema comprueba que el puerto no está en uso.
3. El sistema establece el valor del nuevo puerto.

Escenario de excepción:

2a. El puerto idicado por el usuario está en uso.

    1. El sistema informa del error y finaliza la ejecución del programa.

*a. El usuario aborta la ejecución del caso de uso.

    1. El sistema finaliza la ejecución del programa.

5.1.4.4. Caso de uso: Establecer nivel de registro

Activa: El usuario indica un nivel de registro específico.
Punto de extensión: Registro en caso de uso: Iniciar servidor.
Nivel: Función.
Actor principal: Usuario.
Personal involucrado e intereses:

- Usuario: Requiere establecer un nivel de registro específico para cada prueba de BPELUnit.
- Sistema: Permite establecer un determinado nivel de registro.

Precondiciones: Ninguna.
Postcondiciones: El sistema establece un determinado nivel de registro para la ejecución de las pruebas.
Escenario principal:

1. El usuario indica el nivel de registro de la ejecución de las pruebas. Los más comunes son “ALL” que es el máximo nivel de registro o “NONE”/“OFF” si no se desea obtener ningún registro.
2. El sistema establece el nivel de registro.

Escenario de excepción:

*a. El usuario aborta la ejecución del caso de uso.
    1. El sistema finaliza la ejecución del programa.

5.1.4.5. Caso de uso: Cambiar puerto de BPELUnit

Activa: El usuario indica un puerto específico de ejecución de BPELUnit.
Punto de extensión: Puerto BPELUnit en caso de uso: Iniciar servidor.
Nivel: Función.
Actor principal: Usuario.
Personal involucrado e intereses:

- Usuario: Requiere cambiar el puerto de ejecución de BPELUnit.
- Sistema: Permite cambiar el puerto.

Precondiciones: Ninguna.
Postcondiciones: El sistema cambia el valor del puerto en el que se ejecuta BPELUnit.
Escenario principal:

1. El usuario indica el valor del nuevo puerto de ejecución.
2. El sistema establece el valor del nuevo puerto.

Escenario de excepción:

*a. El usuario aborta la ejecución del caso de uso.
    1. El sistema finaliza la ejecución del programa.

5.1.4.6. Caso de uso: Preparar composición

Actor principal: Sistema.
Personal involucrado e intereses:

- Sistema: Realiza las modificaciones oportunas para que Apache ODE ejecute la composición correctamente.

Precondiciones: Apache ODE está iniciado.
Postcondiciones: El sistema devuelve un archivo zip con los ficheros necesarios para ejecutar la composición de servicios web en Apache ODE.
Escenario principal:

1. El usuario indica la ruta hacia los ficheros BPTS y BPEL.
2. El sistema comprueba que los ficheros existen.
3. El sistema comprueba que existe un fichero deploy.xml en la carpeta padre del fichero BPEL.
4. El sistema crea un archivo zip con el nombre de la composición.
5. El sistema introduce en el zip el archivo deploy.xml.
6. El sistema introduce en el zip el archivo BPEL especificado por el usuario.
7. El sistema busca las dependencias wsdl del proceso BPEL y establece en estos ficheros el puerto del motor y el puerto de BPELUnit adecuados.
8. El sistema introduce en el zip los ficheros wsdl modificados.
9. El sistema busca otras dependencias del proceso BPEL como ficheros xsl o xsd y las incluye en el zip.
10. El sistema genera el zip en la carpeta base de ejecución.

Escenario de excepción:

2a. El fichero BPTS no existe.

    1. El sistema informa del error y finaliza la ejecución.

2b. El fichero BPEL no existe.

    1. El sistema informa del error y finaliza la ejecución.

3a. El fichero deploy.xml no existe en la ubicación del fichero BPEL.

    1. El sistema genera automáticamente el descriptor de despliegue de Apache ODE (deploy.xml) en la carpeta donde se encuentra el fichero BPEL.
    2. Continúa en el paso 4.

*a. El usuario aborta la ejecución del caso de uso.

    1. El sistema finaliza la ejecución del programa.

5.1.4.7. Caso de uso: Parar motor

Actor principal: Sistema.
Personal involucrado e intereses:

- Sistema: Finalizar la ejecución del motor Apache ODE correctamente.

Precondiciones: Apache ODE está iniciado.
Postcondiciones: El sistema termina la ejecución de Apache ODE con éxito.
Escenario principal:

1. El sistema comprueba que el servidor Apache Tomcat no está destruido.
2. El sistema comprueba que el servidor Apache Tomcat no está parado.
3. El sistema para correctamente la ejecución del servidor y con ello la ejecución de Apache ODE.
4. El sistema destruye la instancia de Tomcat.
5. El sistema elimina la carpeta temporal creada (la carpeta base de ejecución).

Escenario de excepción:

1a. La instancia del servidor está destruida.

    1. Continúa con el paso 5.

1a. La instancia del servidor está parada.

    1. Continúa con el paso 4.

*a. El usuario aborta la ejecución del caso de uso.

    1. El sistema finaliza la ejecución del programa.

5.1.5. Modelo conceptual de datos

Mediante el modelo conceptual de datos, obtenemos cuáles son las clases que componen nuestro sistema, así como las relaciones entre las mismas.

En la figura 5.2 podemos ver el diagrama de clases conceptuales de la biblioteca ODE-integration.

Una *ComposiciónWS-BPEL* se ejecuta bajo un *ConjuntoCasosPrueba*. Una vez realizada esta operación, se obtienen los *Resultados* derivados de esta relación de ejecución. Tanto la *ComposiciónWS-BPEL*, el *ConjuntoCasosPrueba* y los Resultados obtenidos poseen un nombre identificador, y una localización en el disco.

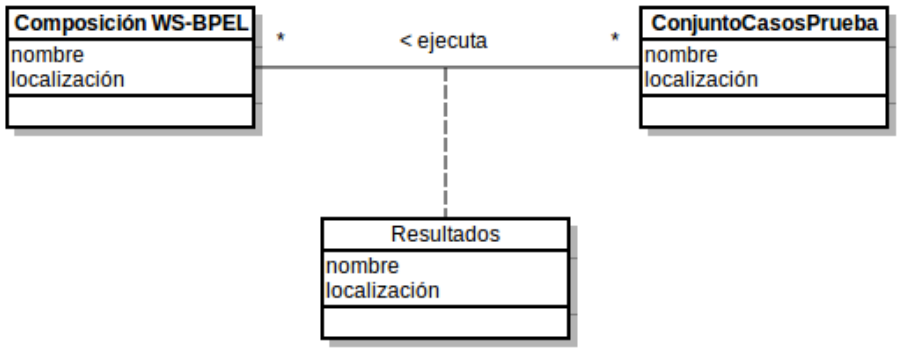

Figura 5.2.: Diagrama de clases conceptuales de ODE-integration

5.1.6. Modelo de comportamiento del sistema

El modelo de comportamiento consta de:

- Diagramas de secuencia del sistema (uno por cada caso de uso descrito en el punto 5.1.4), en los que podemos identificar las operaciones que realiza el sistema.
- Descripción de los contratos para las operaciones del sistema.

5.1.6.1. Probar composición WS-BPEL

En la figura 5.3 podemos ver el diagrama de secuencia del caso de uso Probar composición WS-BPEL.

Operación PrepararMotor(dirBase, puerto, registro)

Responsabilidades Establece las características del motor de ejecución WS-BPEL para ser levantado.

Referencias cruzadas Véase 5.1.4.2.

Precondiciones Ninguna.

Postcondiciones Se crea un objeto de ODEEmbedded ode y se establecen los valores de entrada para los atributos ode.baseDir = *dirBase*, ode.port = *puerto*, ode.fLoggingFilterName = *registro*.

Operación IniciarMotor()

Responsabilidades Levantar el motor Apache ODE.

Referencias cruzadas Véase 5.1.4.2.

Precondiciones Ninguna.

Postcondiciones El sistema inicia Apache Tomcat y con esto se levanta el motor Apache ODE.

Operación refConj := SeleccionarConjuntoCasosPrueba(nombre)

Responsabilidades Selecciona un conjunto de casos de prueba existente.

Referencias cruzadas Véase 5.1.4.6.

Precondiciones Existe un caso de prueba cuyo nombre es igual a *nombre*.

Postcondiciones Se obtiene una referencia *refConj* a un objeto ConjuntoCasosPrueba.

Operación refComp := SeleccionarComposición(nombre)

Responsabilidades Selecciona una composición WS-BPEL existente.

Referencias cruzadas Véase 5.1.4.6.

Precondiciones Existe una composición WS-BPEL cuyo nombre es igual a *nombre*.

Postcondiciones Se obtiene una referencia *refComp* a un objeto ComposiciónWS-BPEL.

Operación refDesc := SeleccionarDescriptorDespliegue(nombre)

Responsabilidades Selecciona un descriptor de despliegue del motor Apache ODE existente.

Referencias cruzadas Véase 5.1.4.6.

Precondiciones Existe un descriptor de despliegue cuyo nombre es igual a *nombre*.

Postcondiciones Se obtiene una referencia *refDesc* a un objeto DescriptorDespliegue.

Operación refZip := PrepararComposición(refComp, refDesc)

Responsabilidades Genera un archivo .zip con los ficheros necesarios para desplegar la composición *refComp* en Apache ODE.

Referencias cruzadas Véase 5.1.4.6.

Precondiciones Existe una referencia a objeto ComposiciónWS-BPEL *refComp* y existe una referencia a objejo DescriptorDespliegue *refDesc*, ambas válidas.

Postcondiciones Se obtiene una referencia *refZip* a un archivo zip generado.

Operación EjecutarComposición(refConj, refZip)

Responsabilidades Ejecuta la composición WS-BPEL respecto a su conjunto de casos de prueba.

Referencias cruzadas Véase 5.1.4.1.

Precondiciones Existe una referencia a un objeto ComposiciónWS-BPEL *refComp* y existe una referencia a un archivo .zip *refZip*, ambas válidas.

Postcondiciones Se obtiene los resultados de la ejecución.

Operación GenerarResultados(salida)

Responsabilidades Genera un fichero XML con los resultados obtenidos en la ejecución.

Referencias cruzadas Véase 5.1.4.1.

Precondiciones Ninguna.

Postcondiciones Imprime los resultados de la ejecución de las pruebas en un documento XML.

Operación PararMotor()

Responsabilidades Parar y destruir el motor Apache ODE.

Referencias cruzadas Véase 5.1.4.7.

Precondiciones Existe una instancia de Apache ODE en ejecución.

Postcondiciones Para el motor Apache ODE y destruye la instancia.

5.1.6.2. Iniciar motor

En la figura 5.4 podemos ver el diagrama de secuencia del caso de uso Iniciar motor.

Las operaciones se encuentran descritas en la sección 5.1.6.1 ya que el caso de uso Probar composición WS-BPEL incluye al caso de uso Iniciar Motor.

Las operaciones de este caso de uso son:

- PrepararMotor(dirBase, puerto, registro)
- IniciarMotor()

5.1.6.3. Cambiar puerto del motor

En la figura 5.5 podemos ver el diagrama de secuencia del caso de uso Cambiar puerto motor.

Operación CambiarPuertoMotor(puerto)

Responsabilidades El usuario indica el puerto de escucha del motor Apache ODE.

Referencias cruzadas Véase 5.1.4.3.

Precondiciones El puerto indicado por el usuario es válido (no está en uso).

Postcondiciones Se informa al sistema del nuevo puerto de escucha del motor.

Operación OperacionesPuertoMotor(puerto)

Responsabilidades El sistema establece el nuevo puerto de escucha del motor Apache ODE.

Referencias cruzadas Véase 5.1.4.3.

Precondiciones Existe una instancia de la clase ODEEmbedded ode.

Postcondiciones Se establece el atributo ode.port = *puerto*.

5.1.6.4. Establecer nivel de registro

En la figura 5.6 podemos ver el diagrama de secuencia del caso de uso Cambiar puerto motor.

Operación EstablecerNivelRegistro(registro)

Responsabilidades El usuario indica el nivel de registro para la ejecución de las pruebas.

Referencias cruzadas Véase 5.1.4.4.

Precondiciones El nivel de registro indicado por el usuario es válido (siendo *ALL* el máximo y *NONE* el mínimo los niveles más comunes).

Postcondiciones Se informa al sistema del nivel de registro.

Operación NivelRegistro(registro)

Responsabilidades El sistema establece el nivel de registro para la ejecución de las pruebas.

Referencias cruzadas Véase 5.1.4.4.

Precondiciones Existe una instancia de la clase ODEEmbedded ode.

Postcondiciones Se establece el atributo ode.fLoggingFilterName = *registro*.

5.1.6.5. Cambiar puerto de BPELUnit

En la figura 5.7 podemos ver el diagrama de secuencia del caso de uso Cambiar puerto motor.

Operación CambiarPuertoBPELUnit(puerto)

Responsabilidades El usuario indica el puerto de escucha de la herramienta BPELUnit.

Referencias cruzadas Véase 5.1.4.5.

Precondiciones El puerto indicado por el usuario es válido (no está en uso).

Postcondiciones Se informa al sistema del nuevo puerto de escucha de BPELUnit.

Operación PuertoBPELUnit(puerto)

Responsabilidades El sistema establece el nuevo puerto de escucha de la herramienta BPELUnit.

Referencias cruzadas Véase 5.1.4.5.

Precondiciones Existe una instancia de la clase ODEEmbedded ode.

Postcondiciones Se establece el atributo ode.bpelUnitPort = *puerto*.

5.1.6.6. Preparar composición

En la figura 5.8 podemos ver el diagrama de secuencia del caso de uso Iniciar motor.

Las operaciones se encuentran descritas en la sección 5.1.6.1 ya que el caso de uso Probar composición WS-BPEL incluye al caso de uso Preparar composición.

Las operaciones de este caso de uso son:

- refConj := SeleccionarConjuntoCasosPrueba(nombre)
- refComp := SeleccionarComposición(nombre)
- refDesc := SeleccionarDescriptorDespliegue(nombre)
- refZip := PrepararComposición(refComp, refDesc)

5.1.6.7. Parar motor

En la figura 5.9 podemos ver el diagrama de secuencia del caso de uso Iniciar motor.

La operacion se encuentra descrita en la sección 5.1.6.1 ya que el caso de uso Probar composición WS-BPEL incluye al caso de uso Parar motor.

La operacion de este caso de uso es:

- PararMotor()

5.2. Diseño

ODE-integration consta de tres fases para completar la ejecución de pruebas unitarias a composiciones WS-BPEL con el motor Apache ODE: la fase de preparación del motor, la fase de preparación de la composición WS-BPEL y la fase de ejecución.

En la figura 5.10 podemos ver un esquema de este proceso.

Preparación del motor El sistema prepara el motor Apache ODE para ser levantado estableciendo el directorio base de ejecución, el puerto de escucha del motor y el nivel de registro para la ejecución de las pruebas. Una vez preparado, el sistema se encarga de levantar Apache ODE.

Preparación de la composición WS-BPEL El sistema recibe como entrada la definición de los casos de prueba y del proceso WS-BPEL que se va a probar. El sistema analiza el proceso y produce los ficheros necesarios para desplegar el proceso WS-BPEL en el motor Apache ODE, empaquetándolos en un fichero zip que se enviará posteriormente a BPELUnit.

Ejecución Con el fichero zip generado en la fase anterior, el proceso WS-BPEL es ejecutado, invocando uno por uno a los casos de prueba. Esta tarea la realiza BPELUnit. Por cada caso de prueba, BPELUnit indicará a Apache ODE que despliegue el proceso (poner en funcionamiento el servicio web). Una vez que el proceso está listo, BPELUnit se conectará al mismo actuando como cliente y ejecutando la operación indicada en el caso de prueba en el puerto correspondiente. Si el proceso realiza una llamada a un servicio externo, BPELUnit recibirá la petición y responderá en función a lo especificado en el caso de prueba. Finalmente Se genera un informe XML con los resultados obtenidos y mensajes intercambiados.

5.3. Implementación y pruebas

En esta sección hablaremos de las herramientas y tecnologías que se han utilizado en el desarrollo de la biblioteca así como las pruebas que se han implementado para asegurar el correcto funcionamiento de la misma.

5.3.1. Herramientas y tecnologías

5.3.1.1. Java

Java [16] ha sido el lenguaje de programación utilizado para desarrollar ODE-integration. Se ha utilizado la versión 1.6.0_27.

Java es un lenguaje de programación de alto nivel orientado a objetos, desarrollado por un grupo de trece personas dirigido por James Gosling, para la empresa Sun Microsystems en el año 1991.

Los objetivos principales de este lenguaje son:

- Cercanía de sus conceptos a los del mundo real.
- Proceso de desarrollo más sencillo y rápido.
- Facilita la reutilización de diseño y códigos.
- Modificaciones, extensiones y adaptaciones más sencillas.
- Sistemas más estables y robustos.

Java también es el lenguaje de desarrollo de las diferentes herramientas del grupo de investigación que utilizaremos en este proyecto como MuBPEL, GAmeraHOM y Rodan.

5.3.1.2. Eclipse

Eclipse [46] es un entorno de desarrollo integrado (IDE, *Integrated Development Environment*) de código abierto, multiplataforma para desarrollar software. Fue diseñado para facilitar la edición, compilación y ejecución de programas, durante la fase de desarrollo. Eclipse es una aplicación gratuita y de código abierto disponible en la red para su descarga.

Se seleccionó Eclipse como IDE de desarrollo, en concreto:

- Distribución: Eclipse Java EE IDE for Web Developers.
- Versión: Juno Service Release 1.

5.3.1.3. Maven

Maven [18] es una herramienta de software para la gestión del ciclo de vida y construcción de proyectos Java. Es un proyecto de Apache Software Foundation cuya licencia es Apache Software License 2.0.

Maven utiliza un Proyecto de Modelo de objetos (POM)[2] para describir el proyecto de software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos. Viene con objetivos predefinidos para realizar ciertas tareas claramente definidas, como la compilación del código y su empaquetado.

Como herramienta de gestión de ciclos de vida, Maven funciona en torno a fases de manera que para pasar a una fase es necesario haber completado con éxito las fases anteriores. Las fases más importantes son:

$ mvn compile compila el proyecto y deja el resultado en target/classes.

$ mvn test compila las pruebas unitarias y las ejecuta.

$ mvn package empaqueta el bytecode resultado de compilar en un .jar.

$ mvn install instala el .jar en el repositorio local de Maven.

$ mvn deploy despliega el .jar en el repositorio remoto de Maven.

Uno de los recursos más útiles de Maven es su soporte para gestión de dependencia: simplemente hay que definir las bibliotecas de las cuales depende la aplicación y Maven las localiza (bien sea en su repositorio local o en uno central), las descarga y las utiliza para compilar el código.

Maven permite la integración con diferentes IDEs entre los que se encuentra Eclipse.

5.3.1.4. Integración continua

La integración continua es un modelo informático propuesto inicialmente por Martin Fowler que consiste en hacer integraciones automáticas de un proyecto lo más a menudo posible para así poder detectar fallos cuanto antes. Entendemos por integración la compilación y ejecución de pruebas de todo un proyecto.

La integración continua nos ofrece una serie de ventajas:

- Los desarrolladores pueden detectar y solucionar problemas de forma continua.

[2] POM es un archivo XML que contiene información sobre los detalles del proyecto y de configuración utilizados por Maven para construir el proyecto. Contiene los valores por defecto para la mayoría de los proyectos.

- Disponibilidad constante de un prototipo para pruebas, demos o lanzamientos anticipados.
- Ejecución inmediata de las pruebas unitarias.
- Monitorización continua de las métricas de calidad del proyecto.

En este proyecto hemos seleccionado Jenkins [47] como herramienta de integración continua.

Jenkins Es una herramienta de integración continua (CI) desarrollada por Kohsuke Kawaguchi, de código abierto escrita en Java. Nace en 2011 y es un proyecto que surge como un fork [3] de Hudson, desarrollado inicialmente en 2004 dentro de Sun Microsystems.

Es una aplicación java web por lo que funciona en un servidor que es un contenedor de servlets, como Apache Tomcat. Soporta herramientas de control de versiones como CVS, Subversion, Git, Mercurial, Perforce y Clearcase y puede ejecutar proyectos basados en Apache Ant y Apache Maven, así como scripts de shell y programas batch de Windows. Además permite ejecutar tareas adicionales previo y posterior a la compilación como preparar el entorno, preparar un emulador, realizar un despliegue o compactar y subir binarios a un FTP.

Algunas de sus características son las siguientes:

- La instalación es sencilla.
- Monitoriza la ejecución de las tareas.
- Se puede configurar completamente desde su interfaz web.
- Genera resultados por correo electrónico para obtener notificaciones en caso de fracasos.
- Permite añadir, borrar, desactivar y activar plugins que extienden la funcionalidad de la herramienta.
- Genera informes para JUnit.
- Posee un historial de cambios realizados por build o versión, de manera que podemos conocer quién ha realizado los cambios, qué ficheros fueron manipulados y sus comentarios al respecto.

[3] El término *fork* se utiliza para denominar la realización de un software tomando como base otro existente y en vistas a cumplir objetivos diferentes o a una ampliación de ellos.

5.3.1.5. Sistema de control de versiones

Un sistema de control de versiones es un sistema de gestión de archivos y directorios, cuya principal característica es que mantiene la historia de los cambios y modificaciones que se han realizado sobre ellos a lo largo del tiempo. De esta forma, el sistema es capaz de "recordar" las versiones antiguas de los datos, lo que nos permite examinar el histórico de cambios o recuperar versiones anteriores de un fichero, incluso aunque haya sido borrado.

El sistema de control de versiones seleccionado en este proyecto es Apache Subversion [48, 49].

Apache Subversion También conocido como SVN[4], es una herramienta de control de versiones de código abierto con licencia Apache. Su objetivo es sustituir y mejorar al conocido CVS (Concurrent Versions System) ampliamente utilizado.

Subversion puede acceder al repositorio[5] a través de redes, lo que le permite ser usado por varias personas que se encuentran en distintos ordenadores. A cierto nivel, la capacidad para que varias personas puedan modificar y administrar el mismo conjunto de datos desde sus respectivas ubicaciones fomenta la colaboración.

Las principales características de SVN son:

- Mantiene versiones no sólo de archivos, sino también de directorios.
- También se mantienen versiones de los metadatos asociados a los directorios.
- Además de los cambios en el contenido de los documentos, se mantiene la historia de todas las operaciones de cada elemento, incluyendo la copia, cambio de directorio o de nombre.
- Atomicidad de las actualizaciones. Una lista de cambios constituye una única transacción o actualización del repositorio.
- Soporte tanto de ficheros de texto como de binarios.
- Mejor uso del ancho de banda, ya que en las transacciones se transmiten sólo las diferencias y no los archivos completos.
- Mayor eficiencia en la creación de ramas y etiquetas.

[4] *svn* es el nombre que usa en la línea de órdenes
[5] Lugar en el que se almacenan los datos actualizados e históricos de cambios, a menudo en un servidor.

5.3.2. Pruebas

Para asegurar el correcto funcionamiento de la biblioteca se han implementado un conjunto de pruebas unitarias. Se han considerado necesarias realizar cinco pruebas para cubrir las funcionalidades de ésta:

- Generar el descriptor de despliegue correctamente. La primera prueba que se diseñó fue comprobar que el sistema genera automáticamente un descriptor de despliegue de Apache ODE correctamente. Para ello se toma una composición WS-BPEL con su descriptor conocidos, se genera un nuevo descriptor mediante la biblioteca y finalmente se comparan para comprobar que se ha realizado la operación correctamente.
- El motor Apache ODE se inicia correctamente. Esta prueba verifica que el motor Apache ODE se ha levantado correctamente. Para ejecutarla se apoya en la estructura de directorios que se genera al iniciarse.
- Ejecución de una composición WS-BPEL indicando la ruta relativa. Esta prueba confirma que BPELUnit ejecutará los casos de prueba especificados mediante la ruta relativa.
- Ejecución de una composición WS-BPEL indicando la ruta absoluta. Esta prueba confirma que BPELUnit ejecutará los casos de prueba especificados mediante la ruta absoluta.
- Ejecución de una composición WS-BPEL mediante el archivo ZIP. La última prueba realiza dos verificaciones importantes en el proceso de ejecución, por un lado comprueba que el archivo ZIP de la composición se genera correctamente y por otro lado comprueba que BPELUnit ejecuta la composición WS-BPEL contra los casos de prueba contenidos en el ZIP.

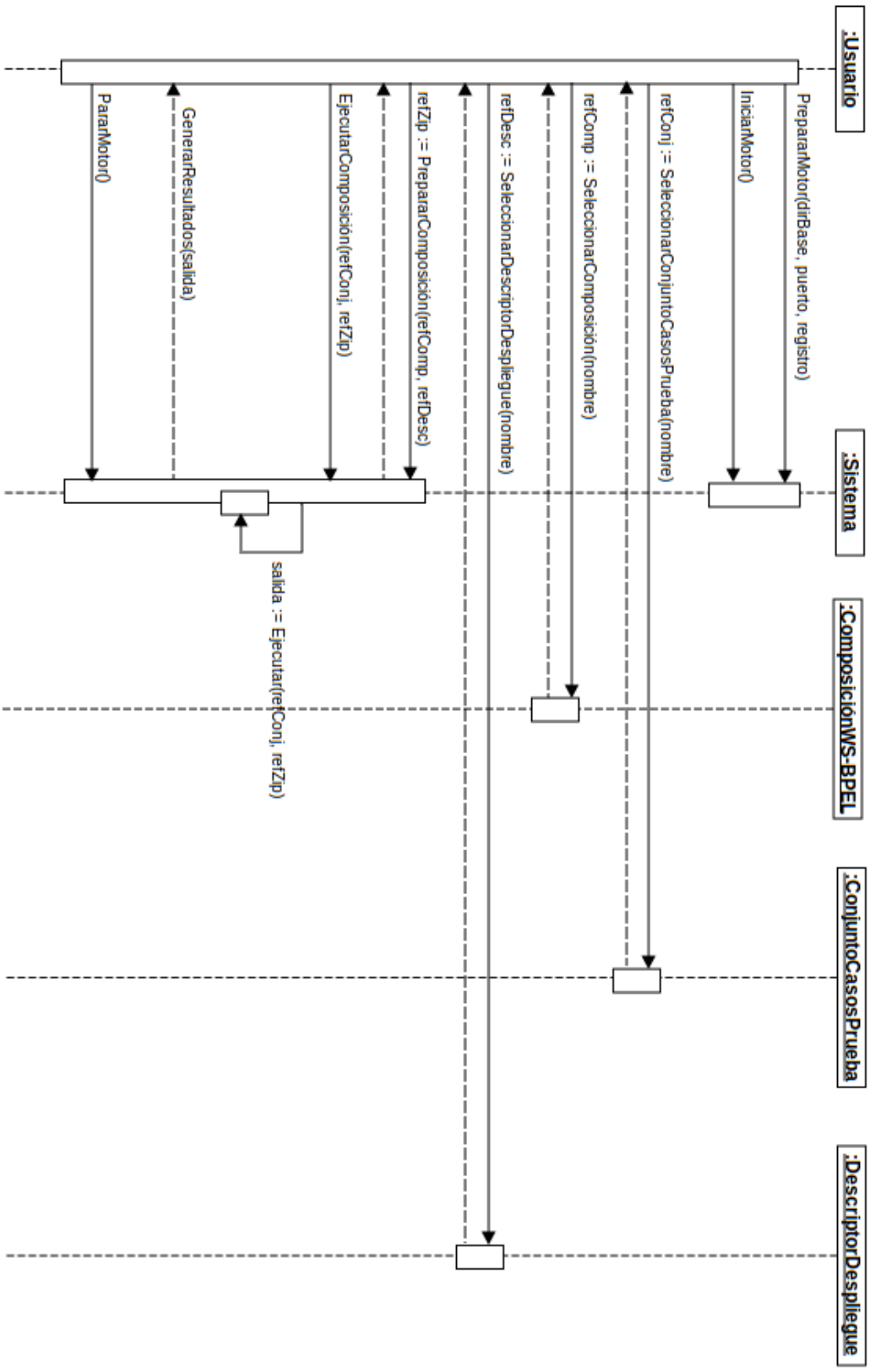

Figura 5.3.: Diagrama de secuencia “Probar composición WS-BPEL”

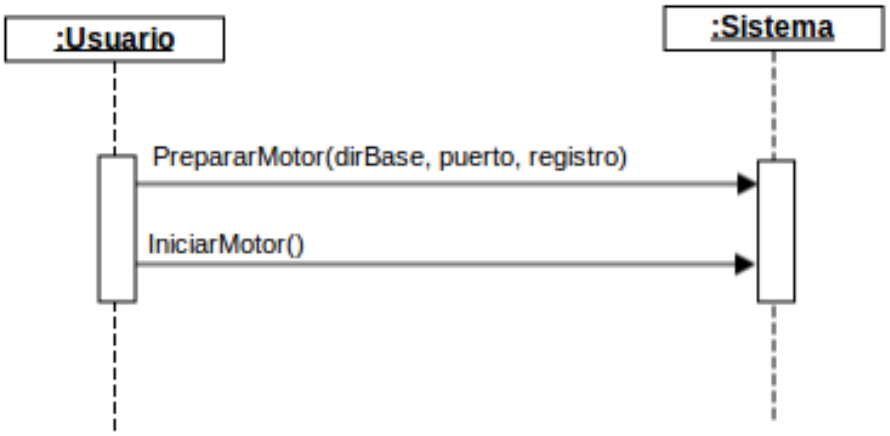

Figura 5.4.: Diagrama de secuencia “Iniciar motor”

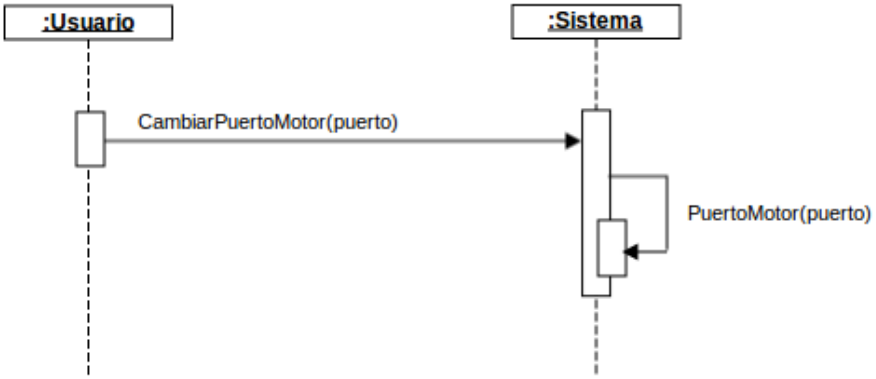

Figura 5.5.: Diagrama de secuencia “Cambiar puerto del motor”

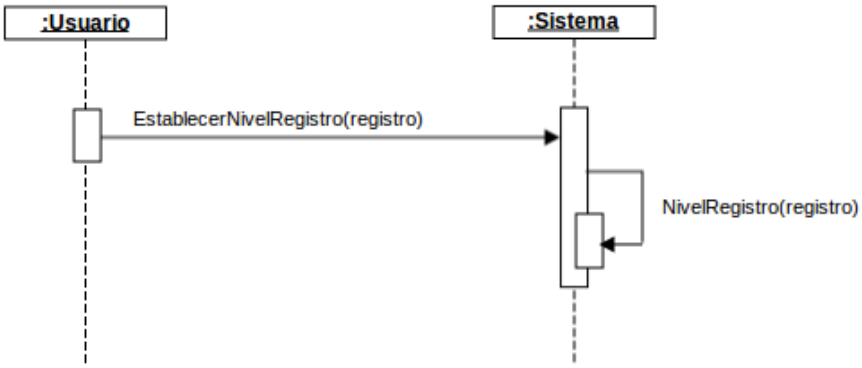

Figura 5.6.: Diagrama de secuencia “Establecer nivel de registro”

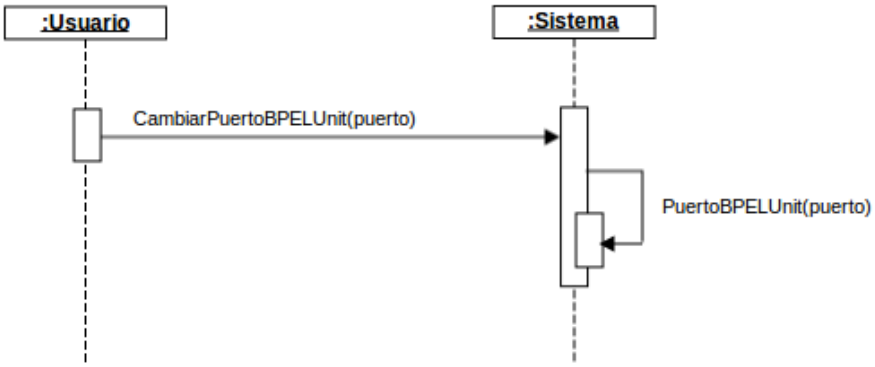

Figura 5.7.: Diagrama de secuencia “Cambiar puerto de BPELUnit”

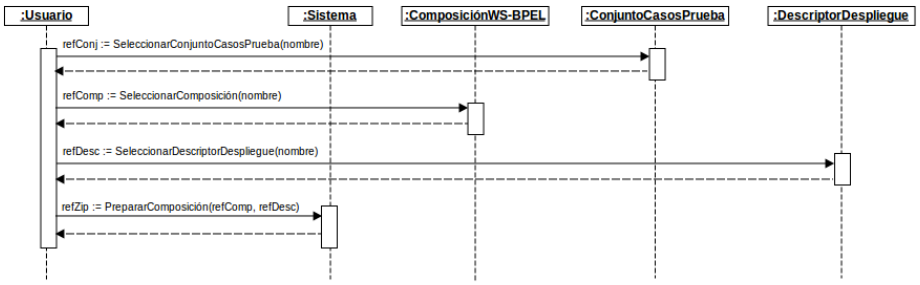

Figura 5.8.: Diagrama de secuencia “Preparar composición”

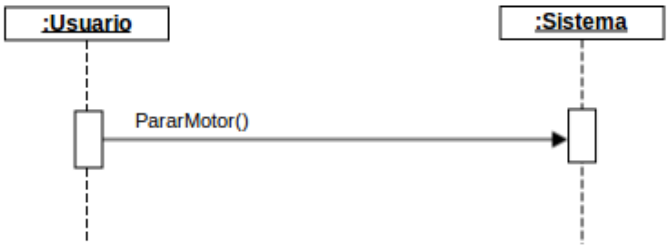

Figura 5.9.: Diagrama de secuencia “Parar motor”

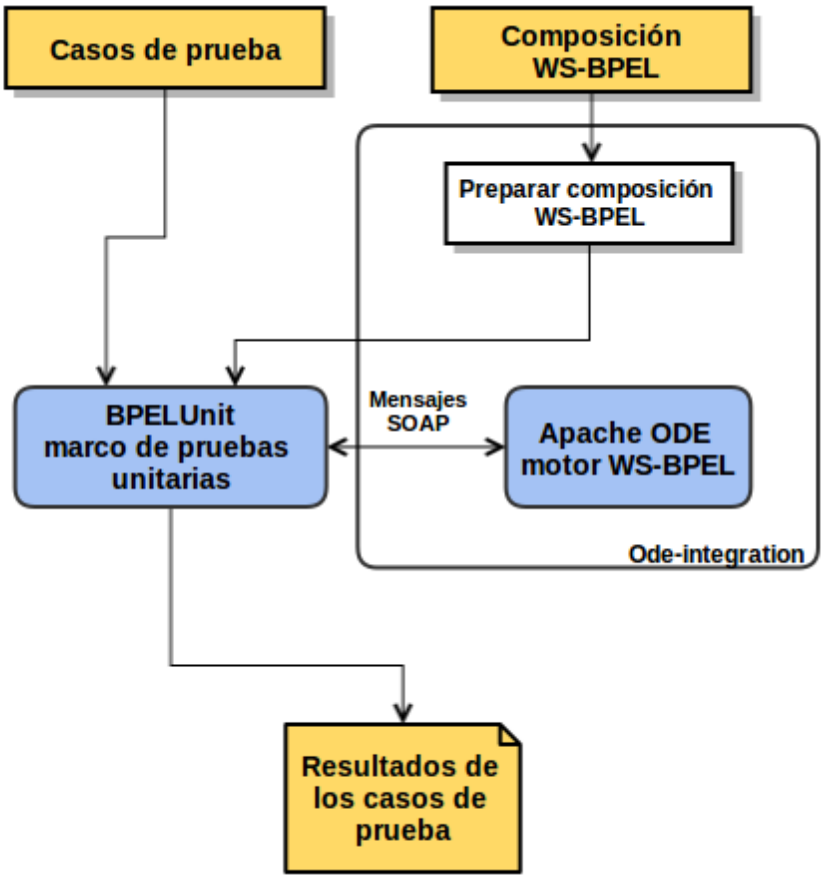

Figura 5.10.: Arquitectura de ODE-integration

6. Integración del motor Apache ODE

En esta sección estudiaremos la arquitectura de las distintas herramientas en las que se integrará el motor Apache ODE, así como la propia integración.

6.1. Arquitectura de MuBPEL

MuBPEL es una herramienta desarrollada por el grupo de investigación UCASE. Permite ejecutar automáticamente la prueba de mutaciones a composiciones WS-BPEL.

Como podemos observar en la figura 6.1, MuBPEL consta de tres módulos: el analizador, el generador de mutantes y el sistema de ejecución. A continuación describiremos cada uno de ellos.

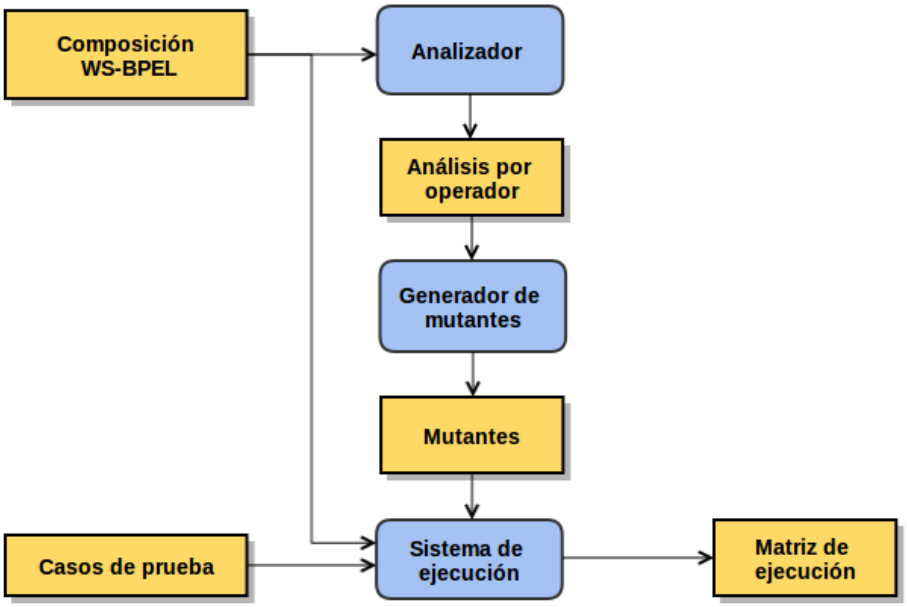

Figura 6.1.: Arquitectura MuBPEL

Analizador El sistema analiza la especificación del proceso BPEL de manera que identifica los elementos de este código original que pueden ser mutados. El sistema devuelve una lista con los operadores que se pueden aplicar al código original

recibido en la entrada. Esta lista está formada por tres componentes, el operador, el operando y el atributo.

- Operador: sirve para identificar al operador que será aplicado a la hora de realizar la mutación.
- Operando: indica en cuantas instrucciones se puede aplicar el operador de mutación.
- Atributo: Representa la mutación que debe realizar el operador en el operando dado.

```
ISV 0 1
EAA 0 4
EEU 0 1
ERR 2 5
ELL 0 1
ECC 0 1
ECN 1 4
EMD 0 2
...
```

Generador de mutantes El sistema genera los mutantes aplicando los operadores de mutación obtenidos en la fase anterior en todos los operandos y en el rango de valores que el valor del atributo indique al código original.

Sistema de ejecución Es aquí donde interviene la biblioteca ODE-integration. Su funcionalidad dentro de MuBPEL es ejecutar las pruebas tanto al programa original como a los mutantes generados en la fase anterior obteniendo los diferentes resultados. Recordemos cómo funciona ODE-integration (figura 6.2):

- El sistema prepara y levanta Apache ODE.
- Genera un archivo zip con todos los ficheros necesarios para desplegar la composición WS-BPEL en el motor.
- BPELUnit ejecuta la composición WS-BPEL mediante los casos de prueba.
- Se genera un informe XML con resultados y mensajes intercambiados.

Por último se realiza la fase de comparación en la que se compara la salida de cada mutante para todos los casos de prueba con la salida del programa original para decidir qué mutante sigue vivo[1] y cuál ha muerto[2]. El operador

[1] Un mutante estará vivo si la salida que genera la ejecución de las pruebas de la composición original es la misma que la del mutante.

[2] Un mutante estará muerto si la salida que genera la ejecución de las pruebas de la composición original es distinta de la del mutante.

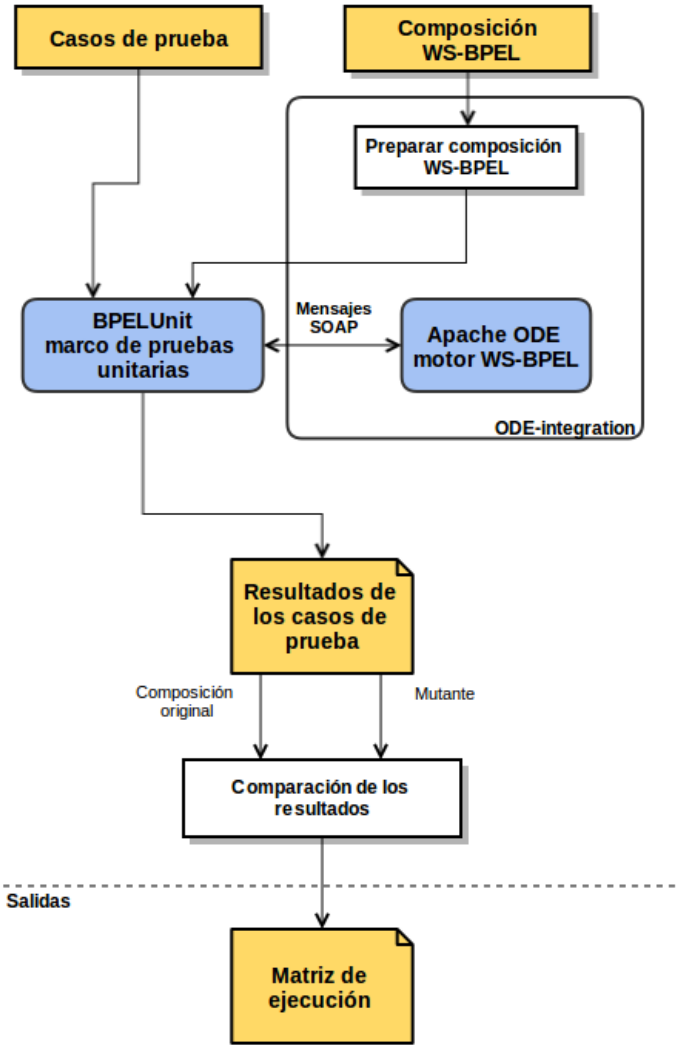

Figura 6.2.: Módulo de ejecución MuBPEL

de comparación compara de forma estricta uno a uno entre los mensajes de respuesta. Con esas comparaciones produce una matriz de ejecución.

6.1.1. Entorno de MuBPEL

La herramienta MuBPEL presenta tres componentes principales:

mubpel Es el componente encargado del análisis de las composiciones WS-BPEL, la generación y la ejecución de los mutantes.

ode-integration Es el componente que encapsula el motor Apache ODE.

bpel-packager Interfaz entre MuBPEL y BPELUnit.

6.2. Arquitectura de GAmeraHOM

GAmeraHOM es una herramienta desarrollada por el grupo UCASE que genera mutantes de orden superior a composiciones WS-BPEL. Como podemos ver en la figura 6.3 la arquitectura de GAmeraHOM es muy parecida a la de MuBPEL, ambas herramientas generan y ejecutan mutantes de composiciones WS-BPEL pero GAmeraHOM tiene una particularidad que lo hace diferente y es que emplea un algoritmo genético para la generación de mutantes de mayor calidad. De esta forma se reduce el coste computacional que supone ejecutar el conjunto completo de mutantes.

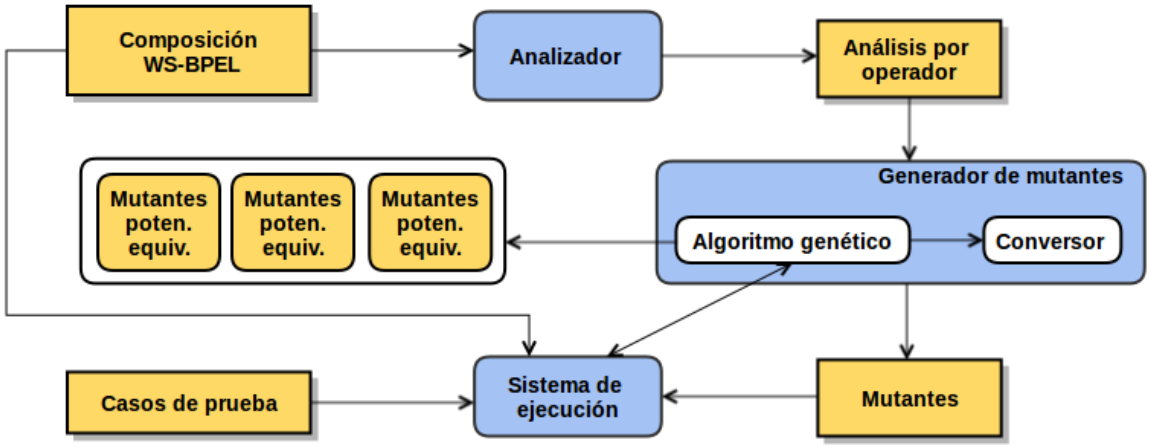

Figura 6.3.: Arquitectura GAmeraHOM

Analizador Se encarga de seleccionar los operadores que pueden ser aplicados a la composición WS-BPEL original que recibe por la entrada.

Generador de mutantes Este módulo difiere de MuBPEL. Está compuesto por un algoritmo genético y un conversor. Ofrece la posibilidad de obtener la población completa de mutantes que se pueden generar a partir de la composición original o bien un subconjunto de ella, aplicando la técnica de la mutación evolutiva mediante el algoritmo genético. Este algoritmo genera nuevos mutantes de orden superior basándose en los resultados de las ejecuciones de los mutantes anteriores, por ello podemos ver en la imagen una relación de retroalimentación con el módulo de ejecución. El conversor generará los mutantes WS-BPEL a partir de los resultados del algoritmo genético empleando hojas de estilo XSLT para realizar las transformaciones pertinentes.

Sistema de ejecución Se encarga de ejecutar tanto la composición WS-BPEL original como los mutantes generados. Es aquí donde actúa la biblioteca ODE-integration,

siendo Apache ODE el motor de ejecución de las composiciones WS-BPEL y BPELUnit el marco de pruebas unitarias a composiciones WS-BPEL.

6.2.1. Entorno de GAmeraHOM

GAmeraHOM es una herramienta genérica que, además de generar mutantes de orden superior, permite que el generador sea extensible y adaptable a cualquier otro lenguaje de programación con la mínima intervención del desarrollador.

GAmeraHOM está dividido en tres componentes relacionados entre sí:

gamerahom-api Modelo de datos e interfaces genéricas.

gamerahom-bpel Soporte de WS-BPEL 2.0 para GAmeraHOM.

gamerahom-core Núcleo genérico y lanzador por línea de órdenes.

Además utilizando el software para la gestión y construcción de proyectos Maven garantizamos que *gamerahom-bpel* sólo dependa de *gamerahom-api* y que *gamerahom-core* no dependa en absoluto de *gamerahom-bpel*.

6.3. Arquitectura de Rodan

Rodan (también denominada GAmeraHOM-ggen) es una herramienta que genera casos de prueba. Al igual que GAmeraHOM utiliza un algoritmo genético para ello. Este algoritmo puntúa mejor a los casos de prueba que maten a los mutantes más difíciles de matar. Rodan genera un conjunto de casos de prueba mejores, que sirven para extender el conjunto original.

En la figura 6.4 podemos observar la arquitectura de la herramienta Rodan.

ServiceAnalyzer Es un analizador de servicios Web, que genera plantillas parametrizadas para producir mensajes de acuerdo a todas las restricciones impuestas desde WSDL, XML Schema y el WS-I Basic Profile 1.1.

TestGenerator Esta herramienta [50] genera datos aleatoriamente y sirven para rellenar las plantillas de ServiceAnalyzer.

Los pasos que sigue esta herramienta para generar los nuevos casos de prueba son los siguientes:

1. ServiceAnalyzer recibe una composición WS-BPEL y genera unas plantillas que deben ser rellenadas.

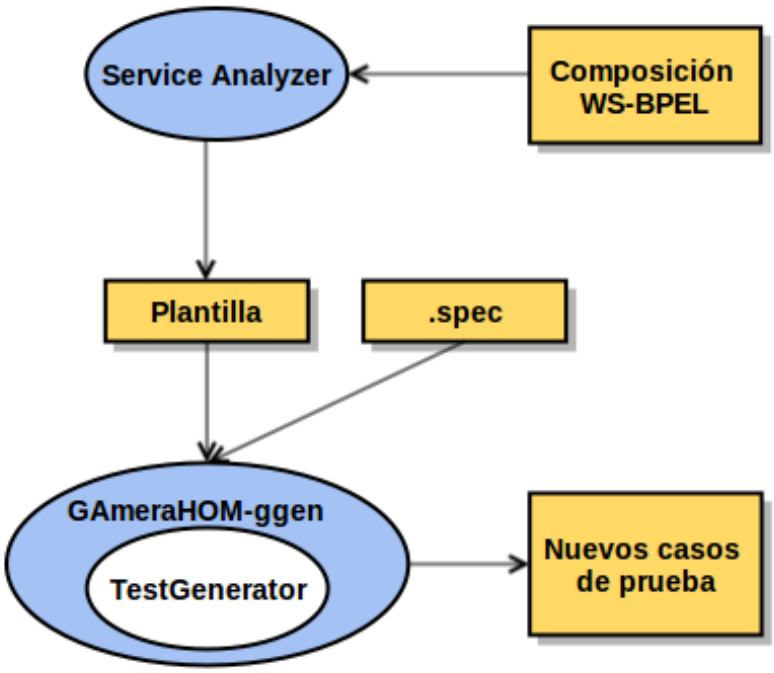

Figura 6.4.: Arquitectura Rodan

2. Se enviará a GAmeraHOM-ggen la plantilla generada en el paso anterior y el fichero con extensión spec. Este fichero sirve para definir el conjunto de datos y restricciones contemplados en ServiceAnalyzer.
3. GAmeraHOM-ggen con ayuda de TestGenerator generará la primera población de casos de prueba aleatorios.
4. Se generan los nuevos casos de prueba mediante el algoritmo genético.

En la fase de generación, se utiliza el motor Apache ODE para ejecutar las composiciones WS-BPEL.

6.3.1. Entorno de Rodan

La herramienta Rodan está formada por los siguientes componentes:

- Los componentes propios de Rodan:

  gamerahom-ggen-api Modelo de datos e interfaces genéricas.

  gamerahom-ggen-core Implementación de la interfaz.

- El componente correspondiente al análisis, generador y ejecución de mutantes:

gamerahom-api Modelo de datos e interfaces genéricas.

gamerahom-bpel Soporte de WS-BPEL 2.0.

gamerahom-core Núcleo genérico y lanzador por línea de órdenes.

- El generador de datos aleatorio (que se usará para generar la primera población):

  test-generator-api Modelo de datos e interfaces genéricas.

  test-generator Implementación de la interfaz.

6.4. Integración de Apache ODE en la herramienta MuBPEL

El motor Apache ODE y la herramienta BPELUnit se necesitan para implementar el módulo de ejecución de MuBPEL y es este módulo el que tendremos que adaptar. Esta adaptación se realizó en tres pasos. En primer lugar se estudió la forma de obtener el motor de manera independiente al código, en segundo lugar se adaptó toda la fase de ejecución con Apache ODE y finalmente se adaptó la fase de comparación para realizarla con Apache ODE.

6.4.1. Localización de Apache ODE

Un requisito en esta integración es que el motor Apache ODE ha de obtenerse independiente al código de MuBPEL de manera que no podemos establecer su dirección grabada “a fuego” dentro del código de la herramienta.

Es por este motivo que se informa a MuBPEL de la localización del motor Apache ODE mediante una propiedad del sistema denominada *mubpel.ode.war*.

Las propiedades del sistema son parejas de clave/valor que definen atributos del entorno de trabajo actual. La clase *System* mantiene un conjunto de estas propiedades de manera que cuando arranca el sistema de ejecución por primera vez, las propiedades del sistema se inicializan para contener información sobre el entorno de ejecución.

Podemos ver algunas de las propiedades predefinidas en la clase System en la tabla 6.1

Tabla 6.1.: Propiedades del sistema predefinidas

Clave	Significado
“file.separator”	Separador de ficheros (p.e., “/”)
“java.class.path”	ClassPath de Java
“java.version”	Número de versión de Java
“line.separator”	Separador de líneas
“os.name”	Nombre del sistema operativo
“user.dir”	Directorio de trabajo del usuario
“user.home”	Directorio “home” del usuario

6.4.1.1. Escribir la propiedad del sistema

MuBPEL se ejecuta desde la consola, mediante un guión en Bash se pasan las opciones necesarias a Java para que funcione, y luego manda todos los argumentos a MuBPEL.

Es en este guión denominado *mubpel* donde definiremos la nueva variable del sistema que indicará a la aplicación la ubicación del motor Apache ODE. Para ello es necesario especificar la propiedad de configuración de la siguiente forma: `-Dpropiedad=valor`.

De este modo nuestro guión lucirá así:

```
1  MUBPEL_DIR=$(dirname $(readlink -f "$0"))
2  JARFILE=$MUBPEL_DIR/${project.artifactId}.jar
3  WARFILE=$MUBPEL_DIR/ode.war
4
5  if [ -z "${JAVA_HOME}" ]; then
6    echo "Please set the JAVA_HOME environment variable to the path to your JVM"
7    exit 1
8  fi
9
10 ${JAVA_HOME}/bin/java \
11   -XX:CompileCommand=exclude,net/sf/saxon/event/ReceivingContentHandler.startElement \
12   -XX:PermSize=128m -Xmx512m -Xss20m \
13   -Dmubpel.ode.war="$WARFILE" \
14   -jar "$JARFILE" $@ \
15   | egrep --line-buffered -v "^CompilerOracle|^### Excluding compile"
```

En la primera línea se localiza el directorio de MuBPEL a partir de la primera orden enviada desde la consola ($0). Seguidamente en la segunda y tercera línea respectivamente localizamos el .jar de la aplicación, así como el fichero ode.war del motor Apache ODE. Finalmente en la línea 13 se establece la propiedad del sistema *mubpel.ode.war* con el par `-Dmubpel.ode.war="$WARFILE"` y en la línea 14 se envían todos los argumentos a MuBPEL.

El resto de argumentos le dan más memoria a MuBPEL y solucionan algunos errores entre bibliotecas.

6.4.1.2. Leer las propiedades del sistema

MuBPEL consta de una clase principal denominada *ExecutionSubcommand* en la que procesa la información introducida por el usuario de manera que según sea la orden que ejecute el usuario, *ExecutionSubcommand* se conecta con las distintas fases de MuBPEL.

Por lo tanto es en esta clase donde recogemos y procesamos la información recibida mediante el guión anterior. Para indicarle a MuBPEL la localización del motor, utilizamos el método predefinido en la clase System de Java *System.getProperty(propiedad)* siendo "propiedad" la propiedad del sistema definida en el guión *mubpel.ode.war*.

6.4.2. Adaptación de la fase de ejecución

Una vez disponible el motor Apache ODE podemos adaptar toda la fase de ejecución de MuBPEL para que funcione con este nuevo motor.

MuBPEL fue desarrollado y probado mediante un gran número de pruebas unitarias realizadas a su código Java (JUnit). En esta ocasión para efectuar la adaptación se realizará el proceso inverso, partiendo de las pruebas unitarias iremos adaptando todo el sistema MuBPEL para que trabaje con Apache ODE.

Las siguientes clases pertenecen al paquete *mubpel* y se encargan de implementar la fase de ejecución de MuBPEL.

- *ExecutionSubcommand*: Clase base de las órdenes de MuBPEL que requieren la ejecución de un proceso WS-BPEL.
- *RunSubcommand*: Implementa la orden *run*.
- *RunSubcommandTest*: Define las pruebas unitarias que se le aplican al código para probar la orden *run* con todas sus opciones.

A partir de las pruebas unitarias definidas en *RunSubcommandTest* se adapta toda la funcionalidad de la fase de ejecución de MuBPEL. Veamos esta funcionalidad.

Ejecución de una composición WS-BPEL con Apache ODE. Se establecen las conexiones necesarias para levantar el motor Apache ODE y enviar la composición a BPELUnit para ser ejecutada.

En primer lugar se prepara el motor Apache ODE para ser levantado en MuBPEL. Esta funcionalidad se añade en clase *CustomizedRunner*.

En segundo lugar se levanta el motor Apache ODE y se envía la composición a probar a BPELUnit para que ejecute las pruebas. Para ello se adapta la clase *RunSubcommand* que se encarga de la ejecución de las pruebas con ActiveBPEL.

Además MuBPEL ofrece una serie de opciones aplicables a la ejecución de las composiciones WS-BPEL que también han sido adaptadas para seguir funcionando con el motor Apache ODE. Veamos estas opciones.

Cambiar el puerto TCP/IP de ejecución del motor y de BPELUnit. Se adaptó MuBPEL para mantener la posibilidad de cambiar el puerto de ejecución de Apache ODE y BPELUnit. Estas opciones se encuentran implementadas en la clase *ODEEmbedded* del paquete *ode-integration* que se encarga de preparar el motor Apache ODE para ser levantado y en la clase *CustomizedRunner* del paquete *bpel-packager* que se encarga de informar a BPELUnit sobre la forma de ejecutar las composiciones.

Establecer todo el seguimiento de MuBPEL. Tenemos como referencia el seguimiento que se hacía con el motor ActiveBPEL:

Se generaba un archivo de registro en la carpeta base de ejecución con todos los comentarios del servidor Jetty. En el caso de Apache ODE generamos un archivo de registro en la carpeta base de ejecución denominado *tomcat.log* que recoge todos los comentarios del servidor Apache Tomcat. Esta funcionalidad está implementada en la clase *ODEEmbedded* del paquete *ode-integration*.

Se generaba un registro por cada caso de prueba en una carpeta denominada "process-logs" en el directorio base de ejecución. Con Apache ODE realizamos la misma funcionalidad obteniendo el mismo resultado (un fichero de registro por cada caso de prueba ejecutado) pero de manera distinta. Para implementar esta utilidad se crea un listener de casos de prueba de BPELUnit que registra y retira los appenders oportunos al principio y final de cada caso de prueba. Está implementado en la clase *LogEachTestCase* en el paquete *bpel-packager*.

Seleccionar el motor que se va a utilizar. La herramienta MuBPEL no sólo puede utilizar el motor que se ha integrado (Apache ODE) sino que también se ha adaptado para ofrecer la posibilidad de utilizar el motor ActiveBPEL desde fuera de la herramienta. Por ello el usuario puede indicar si desea utilizar el motor Apache ODE o ActiveBPEL (por omisión se utilizará Apache ODE).

Ejecutar varias instancias de Apache ODE. Esta ejecución se realiza en paralelo mediante varios hilos de ejecución distintos. En este caso, al haber varios motores ejecutándose se necesitan crear un subdirectorio dentro del directorio base de ejecución, para cada uno de los motores donde se realizará la ejecución normalmente.

6.4.3. Adaptación de la fase de comparación

La adaptación de esta fase se realizó igual que la anterior, partiendo de los casos de prueba y adaptando todo el código para que MuBPEL siga cumpliendo con toda su funcionalidad.

Las pruebas se dividen en dos grupos, el primer grupo comprueban que se realice correctamente la comparación de los resultados de los casos de prueba y el segundo grupo comprueban que se imprime correctamente la matriz de ejecución. Se adaptan las pruebas del primer grupo pues son las que ejecutan las composiciones WS-BPEL.

Las clases principales en la comparación de los resultados en MuBPEL son:

- ExecutionSubcommand: Clase base de las órdenes de MuBPEL que requieren la ejecución de un proceso WS-BPEL.
- RunSubcommand: Implementa la orden *run*.
- CompareSubcommand: Implementa la orden *compare*.
- CompareSubcommandTest: Define las pruebas unitarias que se le aplican al código para probar la orden *compare*.

La adaptación de la fase de comparación de MuBPEL ha requerido los siguientes pasos.

Generar los resultados de la composición original. Para ello se ejecutan las pruebas a la composición original invocando los métodos de la clase *RunSubcommand* la cual pertenece a la fase de ejecución que ya se encuentra adaptada. En este caso tenemos que hacer la llamada correcta pasándole las opciones correspondientes como el puerto de ejecución del motor, el directorio base de ejecución, o la localización del fichero WAR del motor. Está implementado en la clase *CompareSubcommand*.

Generar los mutantes. Para generar estos mutantes se aplican los operadores indicados al código original y posteriormente se crea un fichero en el directorio base de ejecución con la nueva definición del código BPEL mutado. Esta implementación se encuentra en la función *CompareSubcommand*.

Ejecutar los mutantes. Al igual que la composición original, los mutantes se ejecutan invocando a los métodos de la clase *RunSubcommand* de la fase de ejecución. Necesitamos adaptar la llamada a esta función estableciendo las opciones necesarias para poder desplegar y probar la composición WS-BPEL mutada.

Comparar los resultados. Finalmente se comparan los resultados del código original y el mutado imprimiendo la matriz de ejecución en la consola.

6.5. Integración de Apache ODE en las herramientas GAmeraHOM y Rodan

Como hemos visto anteriormente GAmeraHOM y Rodan comparten el componente correspondiente al análisis, generador y ejecución de mutantes. Esto hace que la adaptación de las herramientas se realicen en paralelo pues es este componente el que nos interesa.

En la imagen 6.5 podemos observar la relación de las clases de este componente:

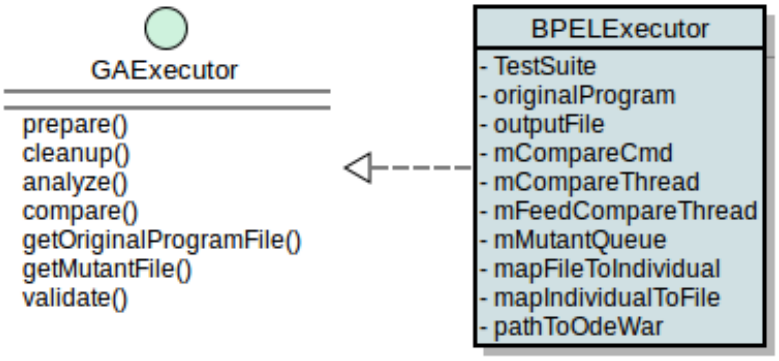

Figura 6.5.: Diagrama de clases del executor

GAExecutor es una interfaz correspondiente al paquete *gamerahom-api* para todos los ejecutores que generan y ejecutan mutantes de un programa en un lenguaje determinado. *BPELExecutor* es una clase correspondiente al paquete *gamerahom-bpel* que implementa la interfaz GAExecutor para el lenguaje WS-BPEL. Esta clase es la que conecta a las herramientas GAmeraHOM y Rodan con MuBPEL.

Para configurar ambas herramientas se utilizan unos ficheros de configuración escritos siguiendo el formato YAML [3] [51].

La nueva versión de MuBPEL requiere la obtención del motor Apache ODE en su distribución WAR para la ejecución de composiciones de servicios web. Por lo tanto realizaremos las siguientes modificaciones para adaptar las herramientas:

- **Descarga del motor automáticamente.** Las herramientas GAmeraHOM y Rodan se han adaptado de manera que, mediante el uso de la herramienta Maven, se descargan el motor Apache ODE en su distribución WAR y se empaqueta de la forma adecuada. Para hacer llegar la ruta del motor que se han descargado las

[3] YAML es un formato de serialización de datos legible por humanos inspirado en lenguajes como XML, C, Python, Perl, así como el formato para correos electrónicos especificado por el RFC 2822. YAML fue propuesto por Clark Evans en 2001, quien lo diseñó junto a Ingy döt Net y Oren Ben-Kiki.

herramientas a MuBPEL, se crea una variable del sistema denominada *"install-dir"* en el guión propio de la herramienta (gamerahom.sh o gamerahom-ggen.sh) que contendrá la dirección de la carpeta base de instalación de la misma. Esta variable del sistema se define en la interfaz *GAExecutor* de manera que se podrá obtener en cualquiera de sus representaciones. En concreto, captamos el valor de la variable en *BPELExecutor* de manera que BPELExecutor busca un archivo ode.war en esta carpeta y se lo indica a MuBPEL para que lo utilice.

- **El usuario especifica el motor de ejecución.** Debido a que las herramientas son genéricas y obtienen el ejecutor en tiempo de ejecución, no es posible implementar directamente la opción para que el usuario introduzca la ruta del motor. En este caso, se han adaptado las herramientas para que el usuario pueda utilizar un motor externo a la aplicación de dos formas: copiando el fichero ode.war (que lo encontrará en la carpeta raiz de la distribución WAR de Apache ODE que se haya descargado) en el directorio de instalación de la herramienta o bien escribiendo la dirección hacia el WAR de Apache ODE en el fichero .yaml que va a utilizar para configurar la herramienta. Esta dirección se introduce en la sección de configuración del *executor* como observamos en el siguiente fragmento:

```
1 executor: !!exec.bpel.BPELExecutor
2   testSuite: LoanApprovalRPC_ODE/loanApprovalProcess.bpts
3   originalProgram: LoanApprovalRPC_ODE/loanApprovalProcess.bpel
4   outputFile: loanApprovalProcess.bpel.out
5   pathToOdeWar: /home/olga/prueba/gamerahom/ode.war
```

Podemos ver un ejemplo completo de un fichero de configuración de la herramienta GAmeraHOM en la sección C.2 y para la herramienta Rodan en la sección D.2.

Con estos cambios las heramientas quedan adaptadas para su uso con Apache ODE.

7. Conclusiones y trabajo futuro

En esta sección presentamos las conclusiones a las que se han llegado tras desarrollar este PFC y el trabajo futuro.

7.1. Valoración

En este PFC se han cumplido todos los objetivos propuestos habiendose realizado:

- Un estudio exhaustivo de los motores WS-BPEL existentes hasta el momento indicando, según las necesidades del grupo, el proceso de selección del motor más adecuado para ellos.
- Un protocolo de adaptación de las composiciones WS-BPEL de ActiveBPEL a Apache ODE. De esta manera el grupo de investigación podrá seguir adaptando el resto de las composiciones que contiene el repositorio para que funcionen con Apache ODE.
- Integración de Apache ODE en MuBPEL. Para ello se ha desarrollado la biblioteca ODE-integration, que envuelve al motor Apache ODE para poder ser integrado en herramientas que ejecuten composiciones de servicios web escritas en el lenguaje WS-BPEL.
- Adaptación de las herramientas GAmeraHOM y Rodan, las cuales están funcionando con el motor Apache ODE.

Además, con el motor Apache ODE obtenemos una mayor separación entre las clases del motor y las clases de la herramienta en la que se ha integrado ya que al utilizar la distribución WAR (*Web Application Archive* - Archivo de Aplicación Web) del motor se ejecuta por separado en cargadores de clases distintos.

La realización de este PFC ha supuesto un reto personal, ya que es el primer proyecto de esta envergadura que he desarrollado.

Ha sido una experiencia enriquecedora tanto personal como profesionalmente ya que gracias a este proyecto he podido experimentar el trabajo de investigación que se realiza en la Universidad de Cádiz colaborando con un gran grupo de profesionales de esta disciplina. Además este grupo hace seminarios cada semana de manera que realizan un seguimiento de los proyectos de sus alumnos colaboradores, donde puedes

presentar los avances que se vayan obteniendo a lo largo del desarrollo del mismo así como preguntar las dudas y problemas que surjan. A su vez, estos seminarios permiten conocer y aprender de los proyectos que realizan los demás compañeros.

Entre los conocimientos que he adquirido desarrollando este proyecto se encuentra el lenguaje de programación Java, ya que en la carrera aprendí C++ como lenguaje orientado a objetos. También aprendí a utilizar el IDE Eclipse para este lenguaje, manejando sus diferentes módulos según la tecnología que se esté desarrollando.

El estudio de los motores WS-BPEL me proporcionó una multitudinaria fuente de conocimientos ya que aprendí lo que es un servicio web, un nuevo lenguaje, WS-BPEL, siendo este lenguaje el que compila y ejecuta el motor seleccionado así como la técnica de prueba de mutaciones. Además obtuve el suficiente criterio como para seleccionar el motor más adecuado de todos los que existían hasta el momento y por supuesto aprendí todas las características que tiene este motor, desde su arquitectura hasta su manejo y manipulación.

Gracias al estudio de las composiciones web conseguí aumentar el conocimiento sobre los lenguajes que se utilizan para desarrollarlas, como el lenguaje WS-BPEL, WSDL, XSLT, XPath o XML Schema y las tecnologías que emplean, como la tecnología SOAP.

El desarrollo de ODE-integration me ha supuesto un aumento del saber en el campo del desarrollo software. Gracias a esta biblioteca he aprendido que es necesario utilizar un sistema de control de versiones para desarrollar proyectos de esta complejidad. En concreto he aprendido a manejar la herramienta Subversion para subir las modificaciones que sufría el proyecto a la forja del grupo UCASE. Además aprendí a utilizar los frameworks de pruebas unitarias JavaUnit y BPELUnit para los lenguajes Java y WS-BPEL respectivamente. Así como a utilizar Jenkins que supervisaba cada modificación de código que realizaba, lanzando alarmas si cometía algún error. También aprendí el uso de la herramienta Maven para crear proyectos así como embeber un servidor de aplicaciones, Tomcat en concreto.

La integración de Apache ODE en MuBPEL así como las adaptaciones de las herramientas GAmeraHOM y Rodan fueron las fases de mayor complejidad del proyecto. En ellas conocí tanto la arquitectura como el uso de estas herramientas, además aprendí a resolver problemas de integración de los diferentes componentes software involucrados, como inconsistencia entre bibliotecas o la conexión entre los diferentes componentes.

Y por último cabe destacar el desarrollo de esta memoria, siendo la documentación de un proyecto de ingeniería y en concreto de un proyecto de investigación se ha mostrado mayor énfasis su calidad.

7.2. Trabajo futuro

El trabajo realizado en este proyecto se puede ampliar de la siguiente forma:

- Automatizando la validación de las composiciones con el estándar WS-I Basic Profile. Se propone desarrollar un nuevo plugin para Eclipse de manera que detecte el error de la composición e indique en que sección del estándar WS-I Basic Profile 1.1 se encuentra descrito.
- Realizando un estudio del impacto del nuevo motor con respecto a los casos de prueba del repositorio. Durante el desarrollo de este proyecto se han detectado indicios de que ODE es más permisivo con las composiciones WS-BPEL en algunos casos que ActiveBPEL, esto puede permitir ejercitar algún operador que con ActiveBPEL no funcionaba bien.

8. Agradecimientos

- A Inmaculada Medina Bulo, por darme la oportunidad de colaborar con el grupo de investigación UCASE.
- A Antonia Estero Botaro y Antonio García Domínguez, por ser mis cotutores en este proyecto, gracias por compartir su conocimiento conmigo.
- A todos los miembros del grupo de investigación UCASE, por su apoyo.
- A mi familia, especialmente a mis padres, por su apoyo, comprensión y confianza a lo largo de estos meses de trabajo.

A. Instalación de Apache ODE en Eclipse

En este punto se describe detalladamente cómo instalar el motor Apache ODE en Eclipse.

- En primer lugar, debemos tener una instancia de Eclipse en nuestra máquina.
  Para ello podemos descargarnos la distribución de Eclipse que queramos en el siguiente enlace `http://www.eclipse.org/downloads/`
  En este caso el tipo de Eclipse seleccionado es Eclipse Java EE IDE for Web Developers y su versión es Juno Service Release 1.
  Una vez descargado, descomprimimos el archivo descargado en la dirección que queramos. Observaremos que se nos ha creado una carpeta llamada "eclipse".
  Para iniciar Eclipse tan sólo hay que pulsar el archivo ejecutable eclipse que se encuentra en su interior.

- A continuación instalamos los plugins en Eclipse necesarios para el lenguaje WS-BPEL: Para ello pulsamos la opción *Help / Install New Software...*. En la ventana nueva introducimos los siguientes datos (Figura A.1):

  Name: BPEL
  Location: http://download.eclipse.org/bpel/site/

  Seleccionamos los 3 plugins (BPEL Commons, BPEL Visual Designer y Runtime Adapter for Apache ODE 1.3) y seguimos las indicaciones para instalarlos (Figura A.2).

- Descargar e instalar Apache tomcat v7
  Vamos a la vista server.
  Pulsamos en *new server wizard* o bien botón derecho *New/Server*.
  Desplegamos la carpeta Apache y encontramos "Tomcat v7.0Server", lo seleccionamos y pulsamos *Next>* (Figura A.3)

  Podemos dejar el nombre que trae por defecto: Apache Tomcat v7.0
  Especificamos el directorio donde lo queremos instalar y pulsamos el botón *Download an Install...* (Figura A.4) Una vez completada la descarga pulsamos *Siguiente* y *Finalizar*.

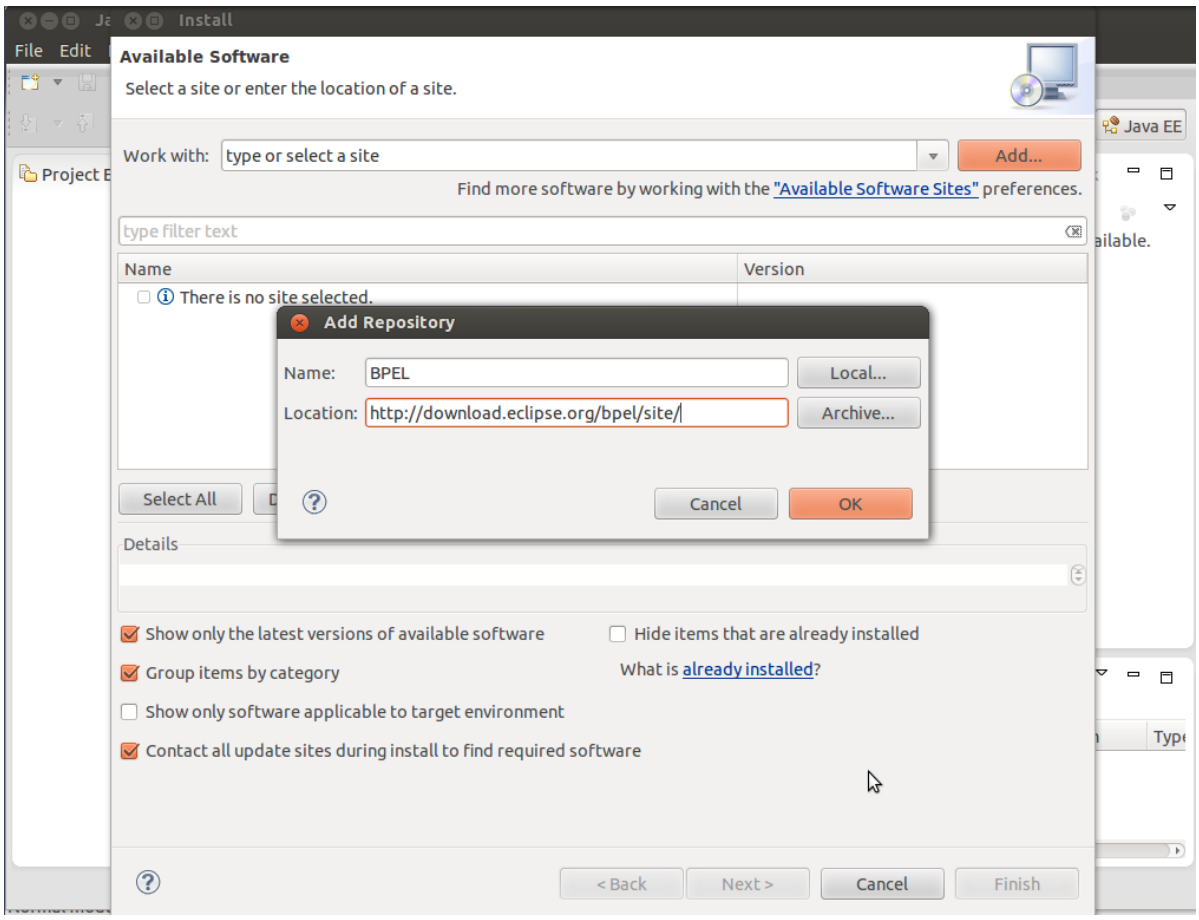

Figura A.1.: Instalación de BPEL en Eclipse (I)

- Instalar Apache ODE
  Abrimos la vista Proyectos, pulsamos botón derecho *Import/WAR File*. Elegimos el archivo ODE.war y aceptamos (Figura A.5).

  A continuación vinculamos el motor ODE con el servidor Tomcat 7:
  Pulsamos en la vista de *Servers* en el servidor Tomcat 7 botón derecho *Add and Remove* Vemos nuestro proyecto ode, lo seleccionamos y pulsamos el botón *Add* (Figura A.6).

- Instalar BPELUnit

  Descargamos BPELUnit desde el site `http://update.bpelunit.net`
  Importamos el site desde *Help/Install New Software*.

  Name: BPELUnit
  Location: http://update.bpelunit.net

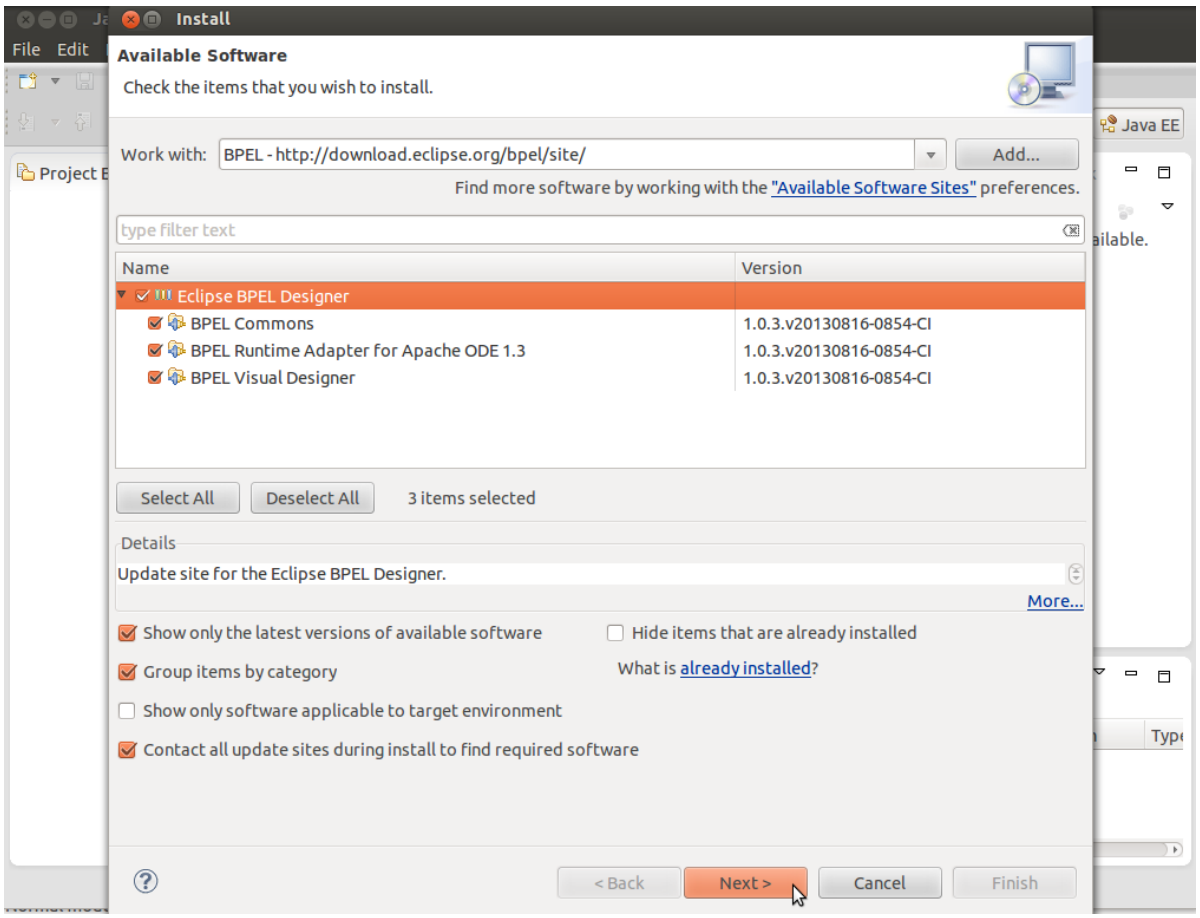

Figura A.2.: Instalación de BPEL en Eclipse (II)

A.1. Tutorial: Hello World. Primera composición con Apache ODE

En este apartado desarrollaremos paso a paso un proceso BPEL sencillo, que se despliega en el motor Apache ODE.

Este tutorial asume que tenemos instalado Eclipse con el motor Apache ODE integrado. Podemos ver un manual de esta instalación en la sección A.

A continuación empezaremos a crear nuestro proyecto.

1. crear un nuevo proyecto BPEL en Eclipse. Para ello seleccionamos *File/New/Project/BPEL 2.0/BPEL Project*. Le damos un nombre al proyecto, en este caso hemos establecido el nombre “HelloWorldBPEL” y pulsamos *Finalizar*. Eclipse nos indica que hay una vista específica para trabajar con BPEL. En este tutorial abriremos esta vista ya que nos ayudará a trabajar con esta tecnología.

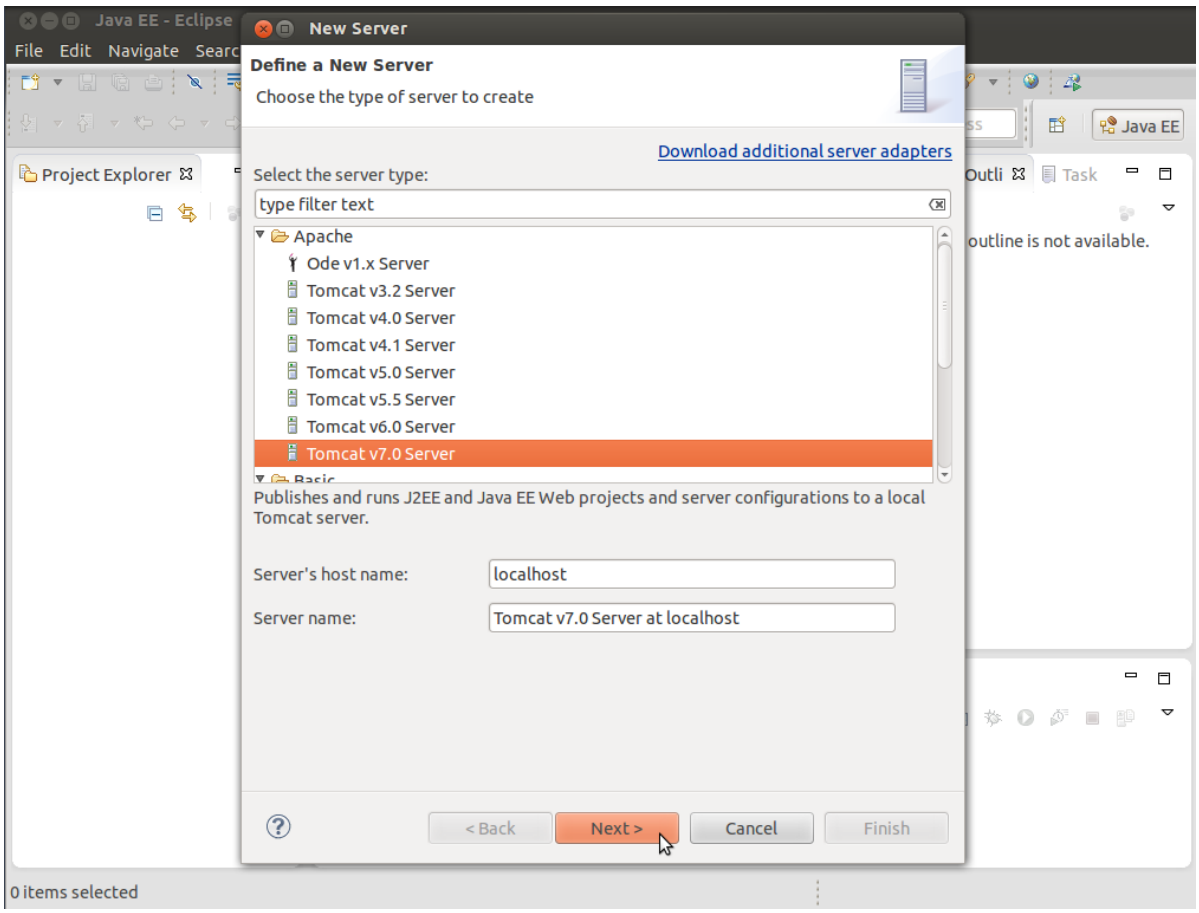

Figura A.3.: Instalación de Tomcat v7 en Eclipse (I)

2. Crear el proceso WS-BPEL. Para ello seleccionamos *File/New/Other/BPEL 2.0/BPEL Process File*. Establecemos el nombre del proceso “HelloWorld” y el namespace “http://helloworld”.

   A continuación establecemos las propiedades de la plantilla como aparece en la imagen A.7.

   En la siguiente pantalla seleccionamos el directorio de la composición WS-BPEL y aceptamos. La estructura del proyecto generado debe de ser:

   - HelloWorldBPEL
     - HelloWorld.bpel
     - HelloWorldArtifacts.wsdl

   Eclipse abre el proceso BPEL en una nueva ventana del Editor. Podemos observar en la imagen A.8 que las actividades aún no se han definido. Este será el siguiente punto.

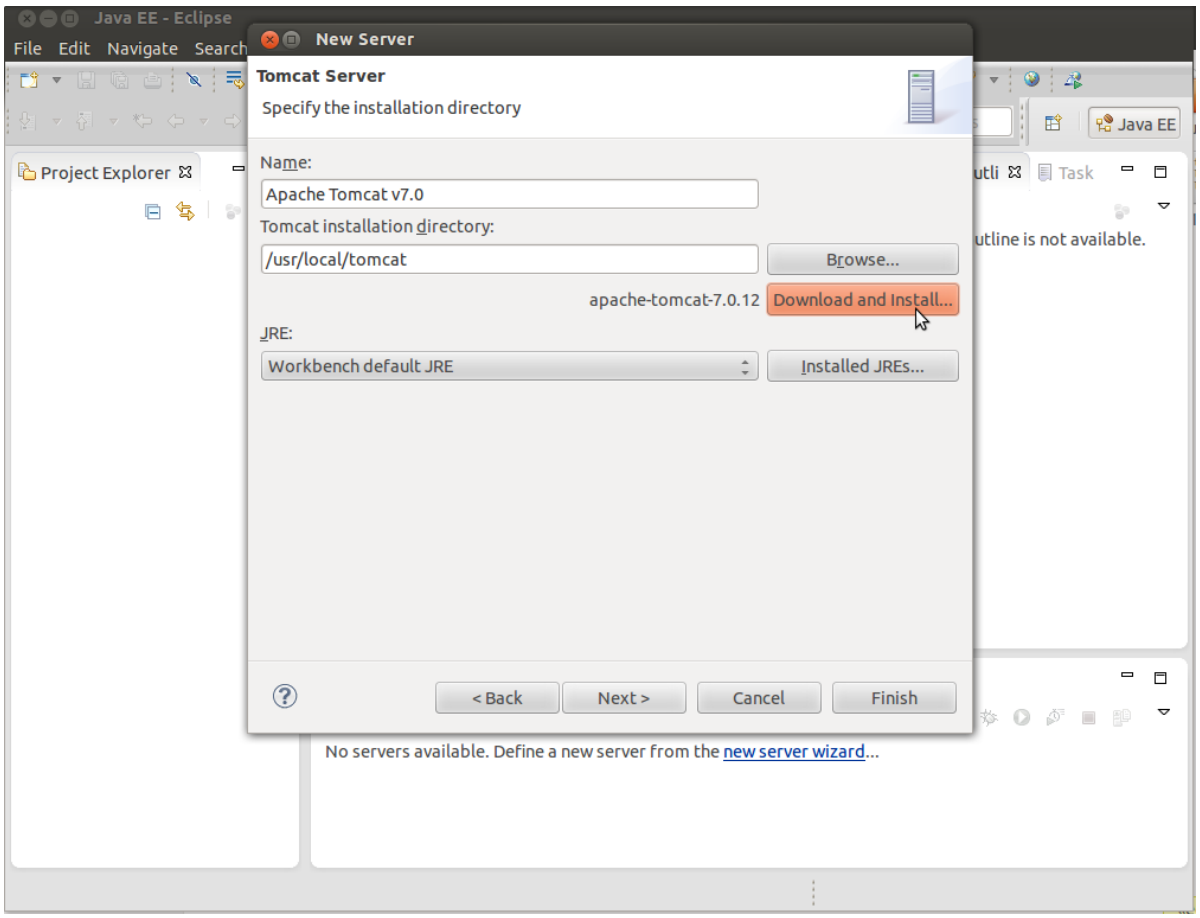

Figura A.4.: Instalación de Tomcat v7 en Eclipse (II)

3. Añadir una actividad Assign. Incluimos la actividad *Assign* entre *receiveInput* y *replyOutput*. Podemos observarlo en la imagen A.9.

4. Establecer las propiedades de esta actividad. Para ello pulsamos con el botón derecho en la actividad *Assign* y seleccionamos *Show in properties*.

   En la pestaña *Details* pulsamos el botón *New* para agregar una nueva acción.

   A continuación realizamos una copia de la variable *input* de la siguiente manera: hacemos doble click en la variable “input ->payload ->result” y se lo asignamos a “output ->payload ->result”.

   Eclipse nos muestra una ventana en la que nos pregunta que si queremos inicializar la variable *output*. Pulsamos *yes* ya que para utilizar las variables primero hay que inicializarlas (imagen A.10). Guardamos los cambios.

5. Crear un servicio en WSDL. Para ejecutar el proceso tenemos que definir un puerto y un enlace (binding) para la interfaz del proceso. Para ello abrimos el fichero “HelloWorldArtifacts.wsdl” en una pestaña nueva del editor haciendo doble click

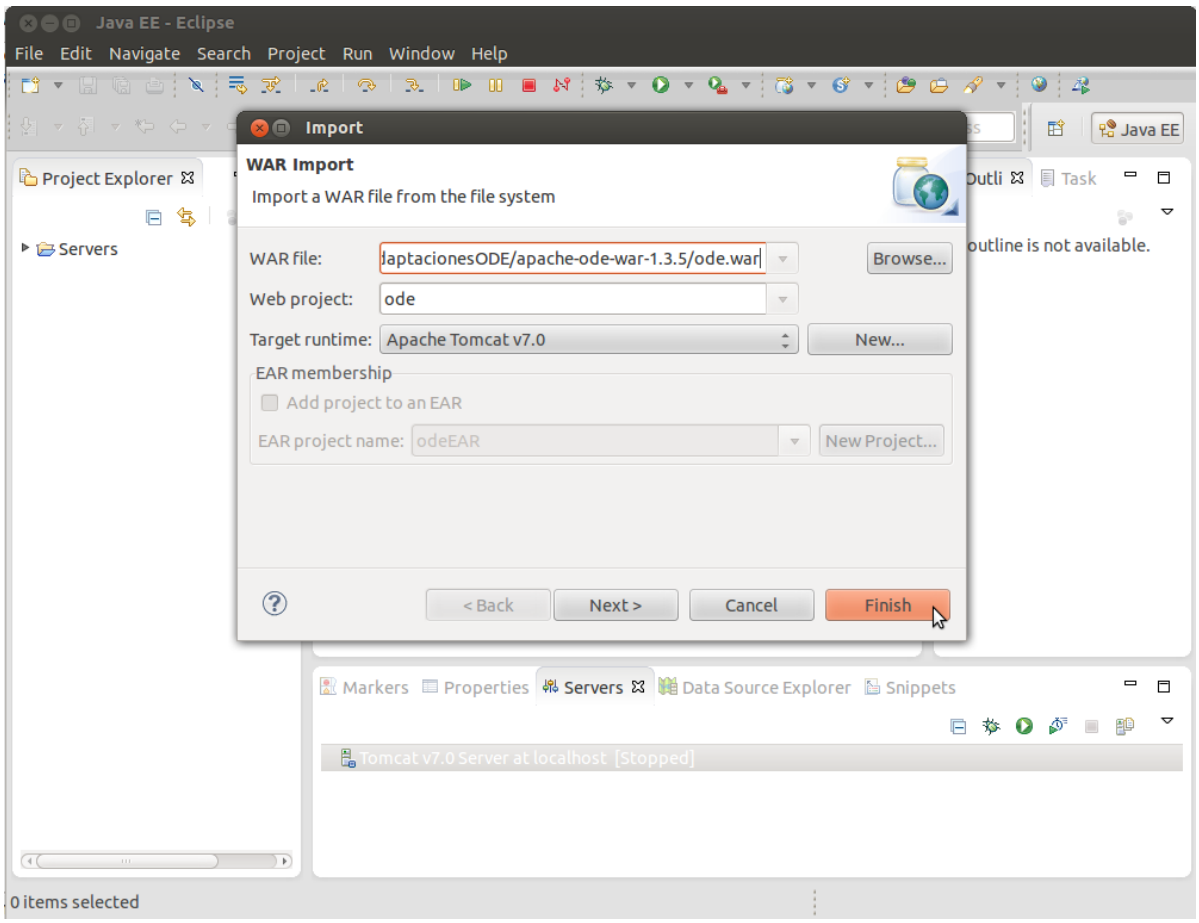

Figura A.5.: Importar Apache ODE en Tomcat v7 (I)

encima del fichero. Se nos abre el editor WSDL. Eclipse genera los datos del servicio gracias a los datos indicados en la imagen A.7. En caso contrario seguir las siguientes indicaciones:

Pulsamos botón derecho *Add Service* y nos aparece en la pantalla el nuevo servicio.

6. Establecer las propiedades del servicio. Especificaremos su nombre pulsando en el servicio con el botón derecho *Show properties* y lo denominaremos “HelloWorldProcessService”. También estableceremos el nombre del puerto pulsando con el botón derecho en el puerto *Show properties* y lo denominaremos “HelloWorldProcessPort”.

7. Especificar el enlace. A continuación tenemos que especificar el enlace o binding del servicio. Para ello pulsamos el botón derecho en cualquier lugar del editor y seleccionamos *Add Binding*. Nos aparecerá el binding en la pantalla. Pulsando sobre él con el botón derecho *Show properties* establecemos el nombre “HelloWorldSOAPBinding”. A continuación establecemos el puerto de conexión pul-

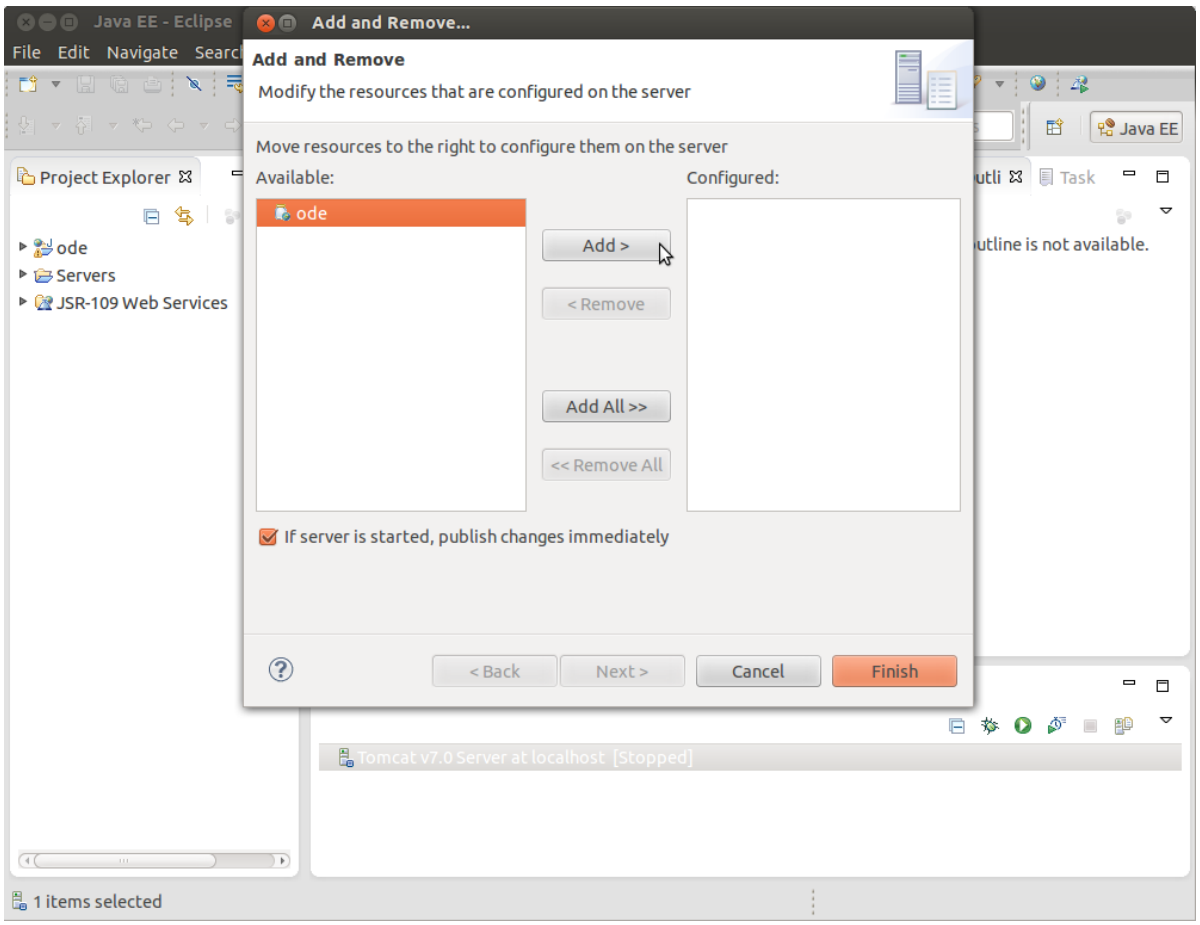

Figura A.6.: Importar Apache ODE en Tomcat v7 (II)

samos con el botón derecho *Set Porttype / Existing Porttype* y seleccionamos el puerto "Hello World". Pulsamos en la vista properties el botón *Generate Binding Content* y seleccionamos el protocolo "SOAP".

8. Por último en las propiedades de "HelloWorldProcessPort" especificamos el binding "HelloWorldSOAPBinding" y la dirección `http://localhost:8080/ode/processes/HelloWorld`.

   El servicio debe ser como el de la imagen A.11.

9. Crear el descriptor de despliegue de Apache ODE. Eclipse contiene una pequeña herramienta que nos facilita crear un descriptor de despliegue para Apache ODE. Para ello pulsamos en *File/New/Other/BPEL 2.0/BPEL Deployment Descriptor* y pulsamos *Finalizar*. Se nos habrá creado en la carpeta "HelloWorldBPEL" el fichero "deploy.xml". Se abrirá este fichero en el editor BPEL Deployment Descriptor.

10. Por último rellenamos la sección "Inbound interfaces" seleccionando el puerto

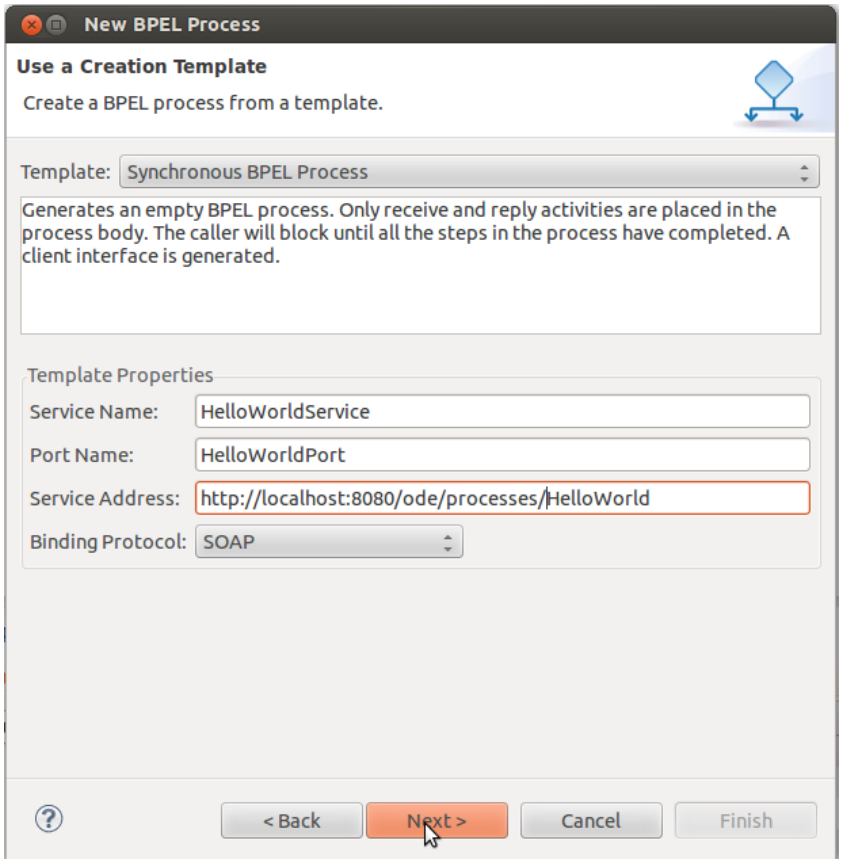

Figura A.7.: Propiedades de la plantilla BPEL

"HelloWorldProcessPort". El resto de las celdas se autocompletarán solas. El deploy final se muestra en la imagen Llegados a este momento, la composición BPEL está completa.

11. Desplegar el proceso. Para poder desplegar el proceso copiamos la carpeta de la composición "HelloWorldBPEL" en la carpeta WebContent/WEB-INF/processes de Apache ODE. Para terminar ejecutamos Tomcat que levantará el motor Apache ODE y éste ejecutará la composición.

Si observamos en la consola de ejecución las siguientes líneas significa que el proceso se ha desplegado correctamente.

```
01:46:42,819 INFO  [BpelServerImpl] Registered process http://helloworldHelloWorld-1.
01:46:42,821 INFO  [DeploymentPoller] Deployment of artifact HelloWorldBPEL successful:
[http://helloworldHelloWorld-1]
```

También podemos comprobar si se ha desplegado el proceso en la interfaz web, accediendo a la dirección `http://localhost:8080/ode/`

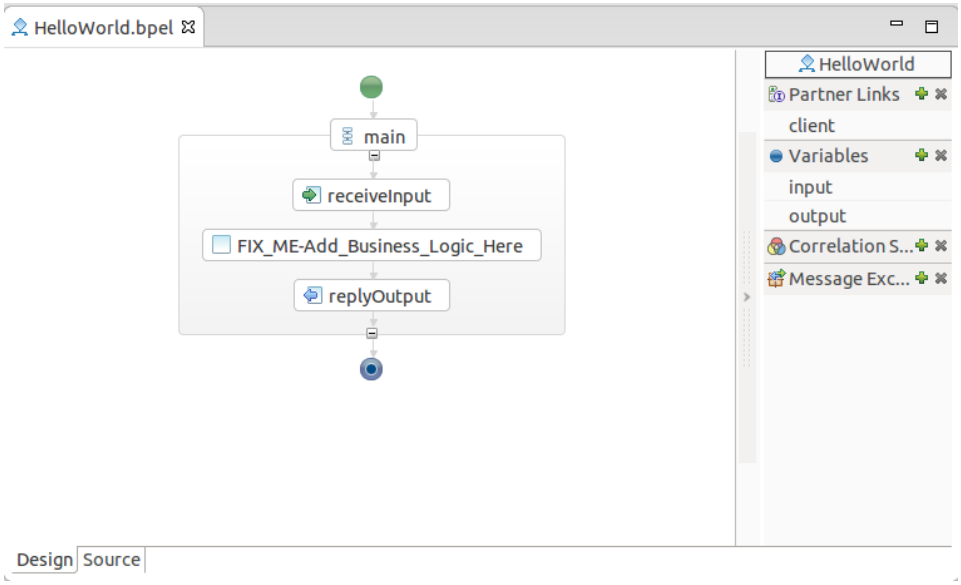

Figura A.8.: Proceso WS-BPEL I

Una vez desplegado el proceso podemos llamarlo de la siguiente manera: `http://localhost:8080/ode/processes/HelloWorld/process?input=HelloWorld`

Obteniendo la siguiente respuesta:

```
<HelloWorldResponse>
    <tns:result>HelloWorld</tns:result>
</HelloWorldResponse>
```

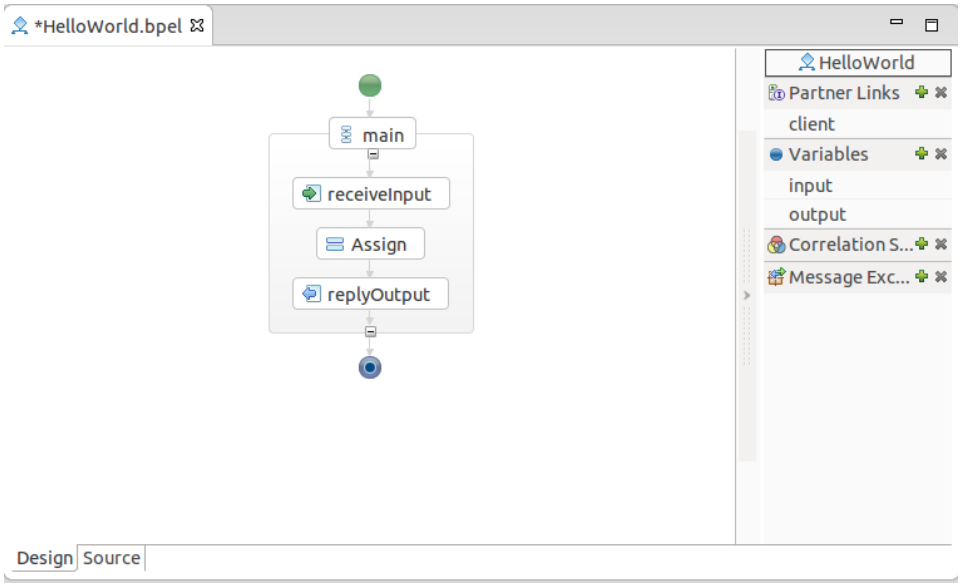

Figura A.9.: Proceso WS-BPEL II

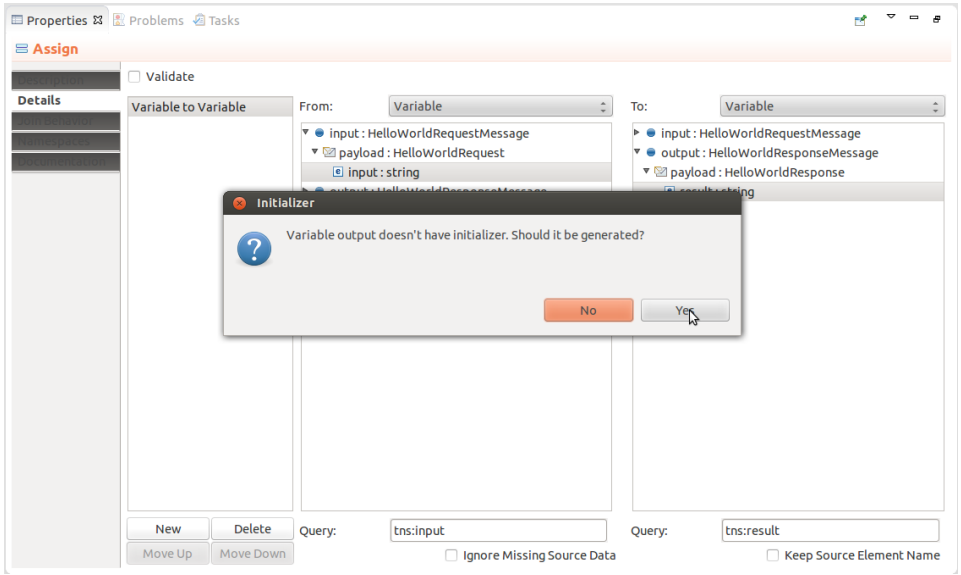

Figura A.10.: Propiedades de la actividad Assign

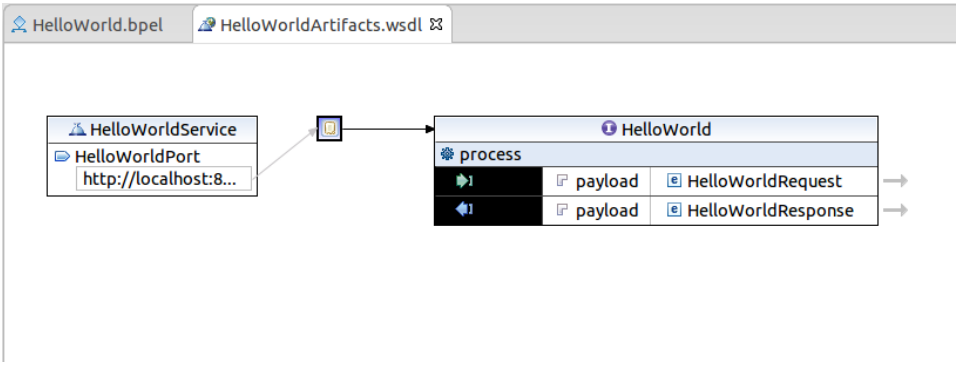

Figura A.11.: Definición del servicio

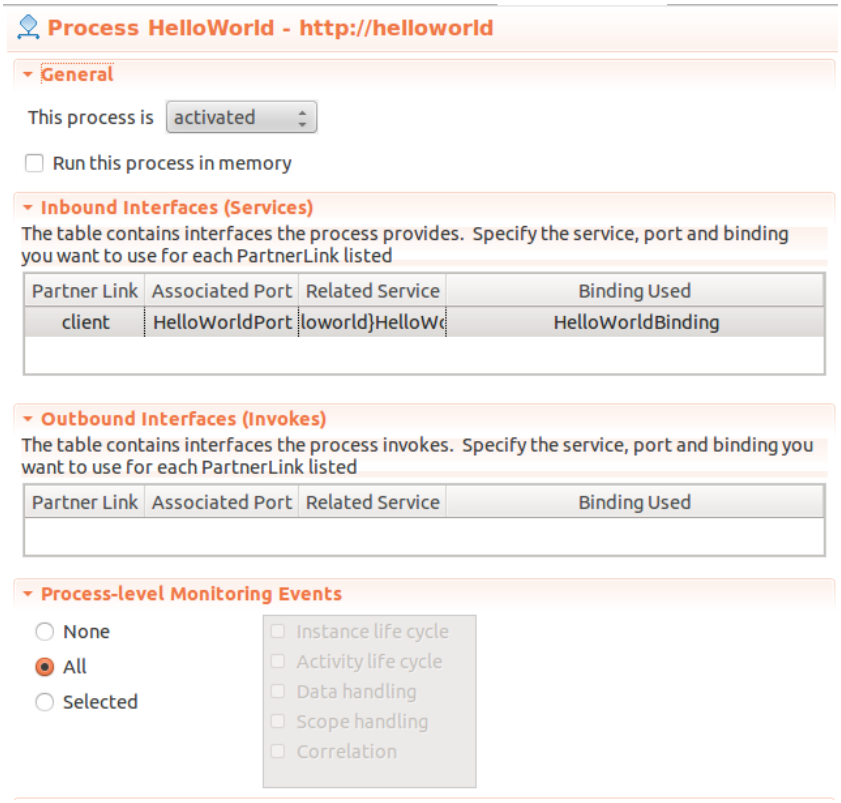

Figura A.12.: Definición del descriptor de despliegue

B. Manual de usuario MuBPEL

En esta sección se presenta el manual de usuario de la herramienta MuBPEL.

B.1. Instrucciones de instalación

MuBPEL se instala mediante un *script* de instalación. Para ello:

- Descargar el script desde: `https://neptuno.uca.es/svn/sources-fm/trunk/scripts/install.sh`
- Ejecutar el script de instalación desde la terminal, de la siguiente forma:

  ```
  bash install.sh mubpel
  ```

- Una vez que se ha terminado la ejecución del script, tenemos que cerrar la sesión y volver a iniciarla.

B.2. Uso de la herramienta

MuBPEL estará listo para su uso, mediante la orden:

```
mubpel (argumentos)
```

A continuación mostraremos los argumentos de MuBPEL mediante la ejecución de la composición LoanApprovalRPC adaptada para Apache ODE.

B.2.1. Analizar una composición WS-BPEL

Esta orden analiza el proceso WS-BPEL y muestra una lista con los operadores que se pueden aplicar a dicho proceso.

```
mubpel analyze bpel|zip
```

Podemos analizar la composición indicando la ubicación del fichero .bpel o bien de la composición WS-BPEL empaquetada en el .zip. Veamos un ejemplo con LoanApproval-RPC.

```
$ mubpel analyze loanApprovalProcess.bpel
ISV 0 1
EAA 0 4
EEU 0 1
ERR 2 5
ELL 0 1
ECC 0 1
ECN 1 4
EMD 0 2
EMF 0 1
AFP 0 1
ASF 1 1
AIS 0 1
AIE 2 1
AWR 0 1
AJC 0 1
ASI 2 1
APM 0 1
APA 0 1
XMF 1 1
XMC 0 1
XMT 0 1
XTF 0 1
XER 0 1
XEE 0 1
AEL 9 1
EIU 1 1
EIN 2 1
EAP 1 1
EAN 1 1
CFA 9 1
CDE 2 2
CCO 2 2
CDC 2 2
```

B.3. Generar y comparar un mutante específico

Si queremos generar un mutante específico usaremos la siguiente orden:

```
mubpel apply bpel|zip operator operandIndex attribute
```

Se creará el mutante resultado de aplicar el operador *operator* a la composición original. Veamos el ejemplo con LoanApprovalRPC.

```
$ mubpel apply loanApprovalProcess.bpel cdc 1 1 > cdc-1-1.bpel
```

Hemos generado el mutante mediante la aplicación del operador CDC, en la primera localización y con el atributo 1. La salida ha sido redireccionada al fichero cdc-1-1.bpel

B.4. Generar todos los mutantes

La siguiente orden genera todos los mutantes de la composición original aplicando los operadores obtenidos en la instrucción anterior.

```
mubpel applyall bpel|zip
```

Veamos el ejemplo con el proceso WS-BPEL.

```
$ mubpel applyall loanApprovalProcess.bpel
```

Los mutantes generados siguen la siguiente nomenclatura:

```
mOO-LL-AA.bpel
```

Donde OO es el índice del operador, LL es el índice de la localización y AA es el índice del atributo.

En este ejemplo veamos los mutantes generados:

```
$ ls m*.bpel
m04-01-01.bpel  m19-01-01.bpel  m30-05-01.bpel
m04-01-02.bpel  m25-01-01.bpel  m30-06-01.bpel
m04-01-03.bpel  m25-02-01.bpel  m30-07-01.bpel
m04-01-04.bpel  m25-03-01.bpel  m30-08-01.bpel
m04-01-05.bpel  m25-04-01.bpel  m30-09-01.bpel
m04-02-01.bpel  m25-05-01.bpel  m31-01-01.bpel
m04-02-02.bpel  m25-06-01.bpel  m31-01-02.bpel
m04-02-03.bpel  m25-07-01.bpel  m31-02-01.bpel
m04-02-04.bpel  m25-08-01.bpel  m31-02-02.bpel
m04-02-05.bpel  m25-09-01.bpel  m32-01-01.bpel
m07-01-01.bpel  m26-01-01.bpel  m32-01-02.bpel
m07-01-02.bpel  m27-01-01.bpel  m32-02-01.bpel
m07-01-03.bpel  m27-02-01.bpel  m32-02-02.bpel
m07-01-04.bpel  m28-01-01.bpel  m33-01-01.bpel
m11-01-01.bpel  m29-01-01.bpel  m33-01-02.bpel
m13-01-01.bpel  m30-01-01.bpel  m33-02-01.bpel
m13-02-01.bpel  m30-02-01.bpel  m33-02-02.bpel
m16-01-01.bpel  m30-03-01.bpel
m16-02-01.bpel  m30-04-01.bpel
```

B.5. Comparar los mutantes con el programa original

En primer lugar es necesario ejecutar la composición original contra los casos de prueba. Para ello se utiliza la siguiente orden:

```
mubpel mubpel run bpts bpel > salida.xml
```

```
$ mubpel mubpel run loanApprovalProcess.bpts
loanApprovalProcess.bpel > salida.xml
```

Hemos redireccionado los resultados de las pruebas al fichero salida.xml.

```
$ head salida.xml
<?xml version="1.0" encoding="UTF-8"?>
<tes:testResult name="Test Suite loanApprovalProcess" result="PASSED"
message="Passed" xmlns:tes="http://www.bpelunit.org/schema/testResult">
    <tes:state name="Status Code">PASSED</tes:state>
    <tes:state name="Status Message">Passed</tes:state>
    <tes:testCase name="Test Case SmallAmountLowRisk" result="PASSED"
    message="Passed">
        <tes:state name="Status Code">PASSED</tes:state>
        <tes:state name="Status Message">Passed</tes:state>
        <tes:partnerTrack name="Partner Track client" result="PASSED"
        message="Passed">
            <tes:state name="Status Code">PASSED</tes:state>
            <tes:state name="Status Message">Passed</tes:state>
```

B.5.0.1. Comparar las salidas hasta la primera diferencia

La siguiente orden compara los resultados hasta la primera diferencia. Una vez que encuentra una diferencia la comparación se detiene y se sigue con el siguiente mutante.

```
mubpel compare bpts bpel xml (bpel1...|-) |
bpts zip xml (zip1...|-)
```

Esta orden devuelve una matriz de ejecución. Cada fila de la matriz representa la comparación de los casos de prueba de un mutante con el programa original de manera que observamos el nombre del mutante, la comparación de los casos de prueba y los tiempos que han gastado en la ejecución.

La solución de la comparación de los casos de prueba puede ser:

- Número 0: Coinciden las salidas de la composición original y del mutante en ese caso de prueba.

- Número 1: Las salidas difieren en ese caso de prueba.
- Número 2: El mutante que no puede ser desplegado correctamente (No puede ser ejecutado).

Veamos un ejemplo con LoanApprovalRPC:

```
$ mubpel compare loanApprovalProcess.bpts
loanApprovalProcess.bpel salida.xml m*.bpel
m04-01-01.bpel 0 0 0 0 0 T 1880969639 647391686 889334461 371681902 412414525
m04-01-02.bpel 1 0 0 0 0 T 20059620550 0 0 0 0
m04-01-03.bpel 1 0 0 0 0 T 20068463945 0 0 0 0
m04-01-04.bpel 1 0 0 0 0 T 20053193421 0 0 0 0
m04-01-05.bpel 0 0 0 1 0 T 630467672 652896642 450167373 20043332048 0
m04-02-01.bpel 1 0 0 0 0 T 20064271026 0 0 0 0
m04-02-02.bpel 1 0 0 0 0 T 20049798708 0 0 0 0
m04-02-03.bpel 1 0 0 0 0 T 20050721663 0 0 0 0
m04-02-04.bpel 1 0 0 0 0 T 20058737338 0 0 0 0
m04-02-05.bpel 1 0 0 0 0 T 20057468745 0 0 0 0
m07-01-01.bpel 0 0 0 0 0 T 347728499 471060670 406828137 285123821 284182164
m07-01-02.bpel 0 0 0 0 0 T 318708472 435077537 425259450 251933017 302179635
m07-01-03.bpel 0 0 0 0 0 T 260853075 566422166 406989532 233019365 228627258
m07-01-04.bpel 1 0 0 0 0 T 20064837855 0 0 0 0
m11-01-01.bpel 1 0 0 0 0 T 269059702 0 0 0 0
m13-01-01.bpel 0 1 0 0 0 T 304448629 293242306 0 0 0
m13-02-01.bpel 0 0 0 1 0 T 472214587 422334214 361264451 140850862 0
m16-01-01.bpel 1 0 0 0 0 T 146441594 0 0 0 0
m16-02-01.bpel 1 0 0 0 0 T 166449796 0 0 0 0
m19-01-01.bpel 0 0 0 0 0 T 280418204 393538454 615143947 247724366 243104800
m25-01-01.bpel 0 0 0 0 0 T 259345064 369852268 446524184 320080988 246215662
m25-02-01.bpel 1 0 0 0 0 T 143673241 0 0 0 0
m25-03-01.bpel 1 0 0 0 0 T 164985163 0 0 0 0
m25-04-01.bpel 1 0 0 0 0 T 128979460 0 0 0 0
m25-05-01.bpel 1 0 0 0 0 T 397360654 0 0 0 0
m25-06-01.bpel 1 0 0 0 0 T 290507124 0 0 0 0
m25-07-01.bpel 0 1 0 0 0 T 360847701 363350956 0 0 0
m25-08-01.bpel 0 0 0 1 0 T 316126310 400702700 561857550 131296306 0
m25-09-01.bpel 1 0 0 0 0 T 20050211542 0 0 0 0
m26-01-01.bpel 1 0 0 0 0 T 20036971886 0 0 0 0
m27-01-01.bpel 1 0 0 0 0 T 20026363254 0 0 0 0
m27-02-01.bpel 1 0 0 0 0 T 20042680367 0 0 0 0
m28-01-01.bpel 0 0 0 0 0 T 324017324 360454429 560521275 255073919 233963485
m29-01-01.bpel 1 0 0 0 0 T 20048717222 0 0 0 0
m30-01-01.bpel 0 0 0 0 0 T 269889208 388864672 379801440 211828680 511619885
m30-02-01.bpel 1 0 0 0 0 T 20053019784 0 0 0 0
m30-03-01.bpel 1 0 0 0 0 T 20048169888 0 0 0 0
m30-04-01.bpel 1 0 0 0 0 T 20052349384 0 0 0 0
m30-05-01.bpel 1 0 0 0 0 T 20047888459 0 0 0 0
m30-06-01.bpel 1 0 0 0 0 T 20050705611 0 0 0 0
m30-07-01.bpel 0 1 0 0 0 T 289428842 20042459311 0 0 0
m30-08-01.bpel 0 0 0 1 0 T 266627899 590592073 377001852 20049051973 0
m30-09-01.bpel 1 0 0 0 0 T 20038529131 0 0 0 0
m31-01-01.bpel 0 0 0 1 0 T 275825158 356929468 329278746 20043678459 0
m31-01-02.bpel 1 0 0 0 0 T 20038723222 0 0 0 0
m31-02-01.bpel 0 1 0 0 0 T 267017683 20111126632 0 0 0
m31-02-02.bpel 1 0 0 0 0 T 20045027650 0 0 0 0
m32-01-01.bpel 0 0 0 1 0 T 278193830 348334381 377928855 20027959917 0
m32-01-02.bpel 1 0 0 0 0 T 20053743099 0 0 0 0
m32-02-01.bpel 0 1 0 0 0 T 235954375 20099586691 0 0 0
m32-02-02.bpel 1 0 0 0 0 T 20033082416 0 0 0 0
m33-01-01.bpel 0 0 0 1 0 T 307023612 334990710 504402486 20049414871 0
m33-01-02.bpel 1 0 0 0 0 T 20039482039 0 0 0 0
m33-02-01.bpel 0 1 0 0 0 T 282250486 20127943860 0 0 0
m33-02-02.bpel 1 0 0 0 0 T 20054336797 0 0 0 0
```

B.5.1. Comparar las salidas de la composición y de los mutantes completas

En este caso aunque se encuentre una diferencia se siguen comparando todos los casos de prueba. La orden es la siguiente:

```
mubpel comparefull bpts bpel xml (bpel1...|-) |
bpts zip xml (zip1...|-)
```

Veamos el ejemplo con LoanApprovalRPC:

```
$ mubpel comparefull loanApprovalProcess.bpts
loanApprovalProcess.bpel salida.xml m*.bpel
m04-01-01.bpel 0 0 0 0 0 T 1718857475 627090987 670783832 423821215 451868425
m04-01-02.bpel 1 1 1 1 1 T 20064616113 20095022351 20031196428 20039336848 20053011905
m04-01-03.bpel 1 1 1 1 1 T 20050198966 20013398441 20162804277 20053635602 20041281869
m04-01-04.bpel 1 1 1 0 0 T 20085355131 20022447471 20136587002 289394409 323850749
m04-01-05.bpel 0 0 0 1 1 T 396124116 456855446 545449668 20046197509 20049135705
m04-02-01.bpel 1 0 0 0 0 T 20057166835 517813964 420371613 272221697 282710881
m04-02-02.bpel 1 0 0 0 0 T 20044707719 388483820 597422901 257053567 260068893
m04-02-03.bpel 1 0 0 0 0 T 20049607146 393321525 443970007 228928343 354436548
m04-02-04.bpel 1 0 0 0 0 T 20037525646 418658059 405662061 229935664 256662595
m04-02-05.bpel 1 1 1 0 0 T 20025930185 20130326492 302738914 304006677 232023777
m07-01-01.bpel 0 0 0 0 0 T 342191350 367424096 380100573 240406129 439286447
m07-01-02.bpel 0 0 0 0 0 T 294145294 356957824 401603304 244227914 254948762
m07-01-03.bpel 0 0 0 0 0 T 312598615 590213742 327535229 273421360 243965034
m07-01-04.bpel 1 1 1 0 0 T 20047810924 20151213269 20115577895 305748168 516537823
m11-01-01.bpel 1 1 1 0 0 T 277502326 202650704 183635358 253505633 238927940
m13-01-01.bpel 0 1 1 0 0 T 294296011 278627336 214215226 561655504 374293404
m13-02-01.bpel 0 0 0 1 1 T 285111955 400044655 481972884 123178654 155603166
m16-01-01.bpel 1 1 1 1 1 T 127480737 195078088 349617224 175202728 149402029
m16-02-01.bpel 1 1 1 0 0 T 129219579 132807879 144742916 233660090 243666177
m19-01-01.bpel 0 0 0 0 0 T 242483208 360139511 611944018 249263891 235532120
m25-01-01.bpel 0 0 0 0 0 T 294498620 344680986 390160717 225520228 228656174
m25-02-01.bpel 1 1 1 1 1 T 136934523 110101782 128453416 132090541 115649342
m25-03-01.bpel 1 1 1 0 0 T 138339241 156767422 132673158 205096400 297607611
m25-04-01.bpel 1 1 1 0 0 T 125051588 146504800 138192642 528015541 251069500
m25-05-01.bpel 1 1 1 0 0 T 200334623 294118410 227867316 230063538 223061050
m25-06-01.bpel 1 0 0 0 0 T 239462571 571207345 348056972 218973447 186005841
m25-07-01.bpel 0 1 1 0 0 T 257652249 269639460 243738817 246151060 409505915
m25-08-01.bpel 0 0 0 1 1 T 270313565 356630056 335649456 114261319 127263181
m25-09-01.bpel 1 1 1 1 1 T 20045360379 20047520564 20046187826 20046913403 20040060118
m26-01-01.bpel 1 1 1 0 0 T 20050341668 20097502480 20134018374 197341015 254141397
m27-01-01.bpel 1 1 1 1 1 T 20047988228 20109782185 20121790067 20045354931 20042561352
m27-02-01.bpel 1 1 1 0 0 T 20038249274 20125140152 233103804 412028160 238387991
m28-01-01.bpel 0 0 0 0 0 T 257785655 282743631 343812234 232890504 219406045
m29-01-01.bpel 1 1 1 0 0 T 20041533220 20149507628 20117375853 241884154 209859577
m30-01-01.bpel 0 0 0 0 0 T 256816877 324257855 331049150 220060598 533540183
m30-02-01.bpel 1 1 1 1 1 T 20044723702 20041058706 20043878563 20052599917 20042679940
m30-03-01.bpel 1 1 1 0 0 T 20042316477 20068925255 20048432773 235659641 201387270
m30-04-01.bpel 1 1 1 0 0 T 20046123117 20043133345 20026285396 426462262 251845644
m30-05-01.bpel 1 1 1 0 0 T 20046222640 20040303032 20041379210 237018484 199692154
m30-06-01.bpel 1 0 0 0 0 T 20038805770 387851350 546597509 209750273 265703205
m30-07-01.bpel 0 1 1 0 0 T 237565271 20037182248 20039312185 251768819 213308829
m30-08-01.bpel 0 0 0 1 1 T 264304772 324911148 372734200 20047093557 20040934411
m30-09-01.bpel 1 1 1 1 1 T 20032914867 20045685632 20033696601 20034214478 20040042455
m31-01-01.bpel 0 0 0 1 1 T 231129460 348243447 338659404 20047240856 20044014325
m31-01-02.bpel 1 1 1 0 0 T 20053169472 20101086603 20088866602 234714062 230708248
m31-02-01.bpel 0 1 1 0 0 T 254709972 20105328969 256395381 224900598 204336872
```

```
m31-02-02.bpel 1 0 0 0 0 T 20034623465 317785173 321143774 265638944 213606632
m32-01-01.bpel 0 0 0 1 1 T 232665622 309504510 510871250 20024265375 20024384739
m32-01-02.bpel 1 1 1 0 0 T 20036144542 20022664402 20130146508 590054227 488696640
m32-02-01.bpel 0 1 1 0 0 T 825758011 20069934168 237819910 208914211 224157343
m32-02-02.bpel 1 0 0 0 0 T 20045499363 334027043 307298225 380536100 255290144
m33-01-01.bpel 0 0 0 1 1 T 256714277 335495601 348126249 20042554074 20050164139
m33-01-02.bpel 1 1 1 0 0 T 20044641429 20110800193 20110135312 208297226 199881494
m33-02-01.bpel 0 1 1 0 0 T 236044190 20114143353 293162075 232749010 235520946
```

B.5.2. Comparar dos salidas de ejecución de una composición

Podemos comparar las salidas de la composición original con uno o más mutantes o bien las salidas entre varios mutantes para ver sus diferencias. Para ello se utiliza la siguiente orden:

```
compareout xml xml1...
```

B.6. Normalizar una composición

Para comparar los mutantes y el programa original es ideal normalizar las composiciones WS-BPEL ya que así conseguimos que todas ellas sigan el mismo estilo, conservando toda su lógica. Para ello se utiliza la siguiente orden:

```
normalize bpel
```

C. Manual de usuario GAmeraHOM

En esta sección se presenta el manual de usuario de la herramienta GAmeraHOM.

C.1. Instrucciones de instalación

GAmeraHOM se instala mediante un *script* de instalación. Para ello:

- Descargar el script desde `https://neptuno.uca.es/redmine/projects/sources-fm/repository/raw/trunk/scripts/install.sh`
- Ejecutar el script de instalación desde la terminal, de la siguiente forma:

  ```
  bash install.sh gamera
  ```

- Seguir las instrucciones para completar la instalación.
- Una vez que se ha terminado la instalación, tenemos que cerrar la sesión y volver a iniciarla.

C.2. Formato de los ficheros de configuración

GAmeraHOM utiliza ficheros YAML para configurar la herramienta.

A continuación veremos un ejemplo completo de un fichero de configuración de GAmeraHOM:

```
1  populationSize: 5
2  maxOrder: &mO 5
3  seed: 42
4
5  executor: !!exec.bpel.BPELExecutor
6    testSuite: LoanApprovalRPC_ODE/loanApprovalProcess.bpts
7    originalProgram: LoanApprovalRPC_ODE/loanApprovalProcess.bpel
8    outputFile: loanApprovalProcess.bpel.out
9    pathToOdeWar: /home/olga/prueba/gamerahom/ode.war
10
11 geneticOperators:
12   !!genetic.OrderMutationOperator {selector: &rndSel !!select.UniformRandomSelection {},
       scaleFactor: &pOMop 0.1, randomRange: *mO} : {probability: *pOMop}
```

```
13   !!genetic.IndividualMutationOperator {selector: *rndSel, scaleFactor: &pIMOp 0.3,
         randomRange: 100} : {probability: *pIMOp}
14   !!genetic.OrderCrossoverOperator {} : {probability: 0.2}
15   !!genetic.IndividualCrossoverOperator {} : {probability: 0.4}
16
17 individualGenerators:
18   !!generate.UniformGenerator {} : {percent: 0.2}
19
20 selectionOperators:
21   !!select.RouletteSelection {} : {percent: 0.4}
22
23 terminationConditions:
24   - !!term.PercentAllMutantsCondition {percent: 0.8}
25   - !!term.GenerationCountCondition {count: 1}
26
27 loggers:
28   - !!log.MessageLogger {console: true, file: }
29   - !!log.HofLogger {console: false, file: hof.txt}
```

Observamos cada uno de sus elementos:

1. Tamaño máximo de la población (línea 1).
2. Orden máximo permitido para los individuos (línea 2). El uso del ampersand, seguido de un nombre, permite que podamos referenciar a ese valor por el nombre de la variable definida.
3. Una semilla determinada para la aleatoriedad (línea 3). El uso de la semilla es opcional, la utilizaremos cuando queramos prefijar la ejecución, obteniendo resultados que se repetibles que facilitarán estudios estadísticos posteriores.
4. Un *executor* encargado de preparar el entorno, limpiar, generar y comparar los mutantes contra el programa original (línea 5). Para ello, tenemos que informarle sobre la ubicación del conjunto de casos de prueba (línea 6), la ubicación del programa original (línea 7), una ruta para un fichero de salida (línea 8) y la dirección hacia el fichero WAR de Apache ODE (línea 9). Esta última sentencia es opcional, ya que si no se especifica el elemento *"pathToOdeWar"* la herramienta utilizará su motor interno.
5. Operadores genéticos que usará el algoritmo, para completar una nueva población cruzando y mutando individuos de la población anterior, previamente seleccionados (línea 11). Cada operador se define dentro del fichero de configuración con la siguiente estructura:

   ```
   !!genetic.NOMBRE {OPCIONES_INTERNAS} : {OPCIONES_COMUNES}
   ```

   Las opciones internas establecidas son:

   - Modo de selección de los individuos que participen en la operación genética. En este caso se utiliza el modo de selección por defecto (método de la ruleta).
   - Valor de escala que representa el mismo valor que la probabilidad de aplicación de ese operador.

- Rango de aleatoriedad que influirán en el nuevo valor resultante de la operación de mutación.

La opción común establecida es:

- Probabilidad de aplicación de cada operador.

6. Generadores de individuos con el porcentaje de individuos que necesitará ejecutar (línea 17). Cada generador se define dentro del fichero de configuración con la siguiente estructura:

   ```
   !!generate.NOMBRE {OPCIONES_INTERNAS} : {OPCIONES_COMUNES}
   ```

   No se necesitan opciones internas pero se deja la estructura preparada, por si en un futuro son necesarias.

7. Operadores de selección de individuos que determinarán de qué forma o siguiendo qué criterios se elegirán un determinado porcentaje de individuos de la población anterior (línea 20). Cada operador de selección se define dentro del fichero de configuración con la siguiente estructura:

   ```
   !!generate.NOMBRE {OPCIONES_INTERNAS} : {OPCIONES_COMUNES}
   ```

   No se necesitan opciones internas pero se deja la estructura preparada, por si en un futuro son necesarias.

8. Criterios de parada para comprobar si el algoritmo debe continuar o no (línea 23). Cada criterio de parada se define dentro del fichero de configuración con la siguiente estructura:

   ```
   !!term.NOMBRE OPCIONES
   ```

   Se dispone de una condición que verifica el número de generaciones y otra que comprueba el porcentaje total de mutantes generados.

9. Mantener los registros. Los *loggers* (registros) nos proporciona información sobre lo que está ocurriendo dentro del algoritmo genético (línea 27). Cada *logger* se define dentro del fichero de configuración con la siguiente estructura:

   ```
   !!log.NOMBRE OPCIONES
   ```

   Existen tres loggers disponibles en GAmeraHOM:

   - MessageLogger: *logger* de mensajes simples.
   - HofLogger: *logger* que muestra los resultados del HOF.
   - NullLogger: *logger* vacío.

   Podemos configurar que estos mensajes sean mostrados bien por consola o bien en un fichero externo. No es posible mostrarlos mediante las dos formas.

C.3. Uso de la herramienta

GAmeraHOM estará lista para su uso, mediante la orden:

```
gamerahom (argumentos)
```

Los argumentos que puede recibir GAmeraHOM son: *–help* y *fichero.yaml*.

C.3.1. Ayuda

```
gamerahom --help
```

Se mostrará la ayuda de la herramienta.

C.3.2. Ejecución de la herramienta

```
gamerahom fichero.yaml
```

El sistema comprobará que el fichero existe y está bien configurado para poder continuar con la ejecución del algoritmo.

Cuando finaliza la ejecución de la herramienta se creará en el directorio de trabajo un fichero *hof.vm*, con los mutantes generados en formato Apache Velocity.

D. Manual de usuario Rodan

En esta sección se presenta el manual de usuario de la herramienta Rodan.

D.1. Instrucciones de instalación

Rodan se instala mediante un *script* de instalación. Para ello:

- Descargar el script desde `https://neptuno.uca.es/svn/sources-fm/trunk/scripts/install.sh`
- Ejecutar el script de instalación desde la terminal, de la siguiente forma:

  ```
  bash install.sh rodan
  ```

- Una vez que se ha terminado la instalación, cerramos la sesión y volver a iniciarla.

D.2. Ficheros de configuración

Rodan utiliza ficheros YAML para configurar la herramienta.

A continuación veremos un ejemplo completo de un fichero de configuración de Rodan:

```
1  populationSize: 6
2  seed: "internationalist"
3
4  executor: !!gamera.exec.bpel.BPELExecutor
5    testSuite: LoanApprovalRPC_ODE/loanApprovalProcess-velocity.bpts
6    originalProgram: LoanApprovalRPC_ODE/loanApprovalProcess.bpel
7    outputFile: loanApprovalProcess.bpel.out
8    pathToOdeWar: /home/olga/prueba/ode.war
9
10 geneticOperators:
11   - !!gamera.ggen.genetic.CrossoverOperator {probability: 0.4}
12   - !!gamera.ggen.genetic.MutationOperator {mutationRange: 10, probability: 0.6}
13
14 individualGenerators:
15   !!gamera.ggen.generate.RandomGenerator {} : {percent: 0.2}
16
17 selectionOperators:
```

```
18   !!gamera.ggen.select.UniformRandomSelection {} : {percent: 0.3}
19   !!gamera.ggen.select.RouletteSelection {} : {percent: 0.7}
20
21 terminationConditions:
22   - !!gamera.ggen.term.PercentAllMutantsCondition {percent: 0.95}
23   - !!gamera.ggen.term.GenerationCountCondition {count: 5}
24   - !!gamera.ggen.term.StagnationMaximumFitness {count: 3}
25   - !!gamera.ggen.term.StagnationAverageFitness {count: 3}
26
27 loggers:
28   - !!gamera.ggen.log.MessageLogger {console: true, file: }
29   - !!gamera.ggen.log.FullHistoryReportLogger {file: history.txt}
30
31 parser: !!testgen.parsers.spec.xtext.integration.XtextSpecParser {spec: LoanApprovalRPC_ODE/
       data.spec}
32
33 formatter: !!testgen.formatters.VelocityFormatter {}
34
35 strategy: !!testgen.strategies.random.UniformRandomStrategy {seed: "0000000000000000"}
36
37 individuals:
38    - [4,1,3]
39    - [7,1,3]
40    - [11,1,1]
41    - [13,2,1]
42    - [16,2,1]
43    - [25,5,1]
44    - [26,1,1]
45    - [27,2,1]
46    - [30,8,1]
47    - [31,1,2]
48    - [32,2,1]
```

Observamos cada uno de sus elementos:

1. Tamaño máximo de la población (línea 1).
2. Una semilla determinada para la aleatoriedad (línea 2). Usaremos esta semilla para generar números pseudoaleatorios.
3. Un *executor* encargado de preparar el entorno, limpiar, generar y comparar los mutantes contra el programa original (línea 4). Para ello, tenemos que informarle sobre la ubicación del conjunto de casos de prueba (línea 5), la ubicación del programa original (línea 6), una ruta para un fichero de salida (línea 7) y la dirección hacia el fichero WAR de Apache ODE (línea 8). Esta última sentencia es opcional, ya que si no se especifica el elemento *"pathToOdeWar"* la herramienta utilizará su motor interno.
4. Operadores genéticos que usará el algoritmo (línea 10). Definiremos la probabilidad de aplicar el operador y una constante de mutación en el caso del operador de mutación.
5. Generadores de nuevos individuos usados (línea 14). Definiremos la probabilidad de generar nuevos individuos usando dicho operador.
6. Operadores de selección de individuos (línea 17). Definiremos cada uno de los operadores de selección a usar, el método de la ruleta y otro aleatorio en este

caso, y la probabilidad de aplicar cada uno de los operadores sobre los individuos de la población.

7. Criterios de parada (línea 21). Definiremos un porcentaje en el caso de tratarse de la condición sobre el porcentaje de mutantes muertos o un valor de cuenta para el resto que nos indicará bien el número de generaciones máximas posible, o el número de generaciones que puede estar sin mejorar o bien el fitness del mejor individuo o el fitness medio de todos ellos.

8. Mantener los registros (línea 27). Los *loggers* (registros) nos proporciona información sobre lo que está ocurriendo dentro del algoritmo genético. Existen dos *loggers* disponibles en Rodan:

   - MessageLogger: *logger* de mensajes simples.
   - HofLogger: *logger* que muestra los resultados del HOF.

   Podemos configurar que estos mensajes sean mostrados bien por consola o bien en un fichero externo. No es posible mostrarlos mediante las dos formas.

9. Parámetros usados de TestGenerator.

   - Parser: es el componente que lee las variables a rellenar y sus tipos (línea 31).
   - Formatter: es el componente que se indica el formato en el que se volcaran los datos en un fichero(línea 33).
   - Generador a usar: Definiremos el generador uniforme (línea 35).

10. Elección de los mutantes concretos sobre los que se ejecutaran el conjunto de casos de prueba generado (línea 37). En caso de no especificar esta opción, se ejecutarán los casos de prueba contra todos los mutantes. El formato en el que hay que indicar los mutantes es el siguiente: cada fila se corresponderá con un mutante, coincidiendo el primer parámetro con el operador, el segundo con la instrucción y el tercero con el atributo.

D.3. Uso de la herramienta

Rodan estará listo para su uso, mediante la orden:

```
gamerahom-ggen (argumentos)
```

Los argumentos que puede recibir Rodan son: *–help* y *fichero.yaml*.

D.3.1. Ayuda

```
gamerahom-ggen --help
```

Se mostrará la ayuda de la herramienta.

D.3.2. Ejecución de la herramienta

```
gamerahom-ggen fichero.yaml
```

El sistema comprobará que el fichero existe y está bien configurado para poder continuar con la ejecución del algoritmo.

Cuando finaliza la ejecución de la herramienta se creará en el directorio de trabajo un fichero *hof.vm*, con los casos de prueba generados en formato velocity. Además si se especificó en el fichero de configuración capturar los loggers de la herramienta en un fichero externo, también lo encontraremos en el directorio.

Bibliografía

[1] OASIS. Web Services Business Process Execution Language Version 2.0. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html, 2007. Última visita: 11-03-2014.

[2] A. Estero Botaro, F. Palomo Lozano y I. Medina Bulo. Mutation operators for WS-BPEL 2.0. ICSSEA 2008: Proceedings of the 21th International Conference on Software and Systems Engineering and their Applications, 2008.

[3] A. Estero-Botaro, J. Boubeta-Puig, V. Linero-Barea y I. Medina-Bulo. Operadores de mutación de cobertura para WS-BPEL 2.0. En *JISBD'12: Actas de las XVI Jornadas de Ingeniería del Software y Bases de Datos*. 2012.

[4] A. García-Domínguez, A. Estero-Botaro, J. J. Domínguez-Jiménez, I. Medina-Bulo y F. Palomo-Lozano. MuBPEL: una herramienta de mutación firme para WS-BPEL 2.0. En *JISBD'12: Actas de las XVI Jornadas de Ingeniería del Software y Bases de Datos*. 2012.

[5] Grupo UCASE, Universidad de Cádiz. GAmeraHOM. https://neptuno.uca.es/redmine/projects/gamera/wiki, 2011. Última visita: 11-03-2014.

[6] J. Domínguez-Jiménez, A. Estero-Botaro, A. García-Domínguez y I. Medina Bulo. Evolutionary mutation testing. *Information and Software Technology*, **53**(10):1108–1123, 2011.

[7] Á. Galán-Piñero. Generador de casos de prueba genético. http://rodin.uca.es/xmlui/handle/10498/14877, 2012. Última visita: 11-03-2014.

[8] Free Software Foundation. General Public License Version 3. http://www.gnu.org/licenses/gpl.html, 2007. Última visita: 11-03-2014.

[9] The Apache Software Foundation. Apache Software License Version 2.0. http://www.apache.org/licenses/LICENSE-2.0, 2004. Última visita: 11-03-2014.

[10] W3C. World Wide Web Consortium. Working Group Note. http://www.w3.org/TR/ws-arch/, 2004. Última visita: 11-03-2014.

[11] OASIS. Reference Model for Service Oriented Architecture 1.0. http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html, 2006. Última visita: 11-03-2014.

[12] W3C. SOAP Recomendation Version 1.2. http://www.w3.org/TR/soap12-part1/, 2007. Última visita: 11-03-2014.

[13] W3C. XML Path Language (XPath) Version 1.0. http://www.w3.org/TR/xpath/, 1999. Última visita: 11-03-2014.

[14] W3C. XSL Transformations (XSLT) Version 1.0. http://www.w3.org/TR/xslt, 1999. Última visita: 11-03-2014.

[15] BPELUnit. http://bpelunit.github.io/, 2014. Última visita: 11-03-2014.

[16] Oracle. Java Runtime Environment. http://java.com/es/download/whatis_java.jsp, 2014. Última visita: 11-03-2014.

[17] Sun Microsystems. OpenJDK. http://openjdk.java.net/, 2014. Última visita: 11-03-2014.

[18] T. O'Brien, J. van Zyl, B. Fox, J. Casey, J. Xu y T. Locher. *Maven: By example. An introduction to Apache Maven*. Sonatype, 2010. ISBN 978-0984243334.

[19] Web Services-Interoperability Organization (WS-I). Basic Profile Version 1.1. http://www.ws-i.org/profiles/basicprofile-1.1-2004-08-24.html, 2004. Última visita: 11-03-2014.

[20] W3C. Web Services Description Language (WSDL) Version 1.1. http://www.w3.org/TR/wsdl, 2001. Última visita: 11-03-2014.

[21] OASIS. Web Services Security (WSS) Version 1.1. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss, 2006. Última visita: 11-03-2014.

[22] OASIS. WS-BPEL Extension for People (BPEL4People) Specification Version 1.1. http://docs.oasis-open.org/bpel4people/bpel4people-1.1.html, 2010. Última visita: 11-03-2014.

[23] Informatica. Communicating with the ActiveBPEL Server Administration Interface via Web Services. http://www.activevos.com/content/developers/education/sample_active_bpel_admin_api/doc/index.html, 2008. Última visita: 11-03-2014.

[24] Informatica. Active VOS. http://www.activevos.com/, 2014. Última visita: 11-03-2014.

[25] Apache Incubator. Apache Agila. http://incubator.apache.org/agila/, 2005. Última visita: 11-03-2014.

[26] The Apache Software Foundation. Apache ODE. http://ode.apache.org/, 2013. Última visita: 11-03-2014.

[27] The Apache Software Foundation. Apache ServiceMix. http://servicemix.apache.org/index.html, 2014. Última visita: 11-03-2014.

[28] The Apache Software Foundation. Apache Tuscany. http://tuscany.apache.org/, 2012. Última visita: 11-03-2014.

[29] Microsoft Integration. BizTalk Server. http://www.microsoft.com/biztalk/en/us/default.aspx, 2013. Última visita: 11-03-2014.

[30] Bpel-g group. Bpel-g. http://code.google.com/p/bpel-g/, 2012. Última visita: 11-03-2014.

[31] Binary Spectrum. eInsight Business Process Manager. http://www.binaryspectrum.com/service-oriented_architecture/eInsightBusinessProcessManager.html. Última visita: 05-02-2014.

[32] Red Hat. JBoss Fuse. http://www.redhat.com/products/jbossenterprisemiddleware/fuse/, 2014. Última visita: 11-03-2014.

[33] Apache Geronimo Application Server. GASwerk. http://gaswerk.sourceforge.net/index.html, 2008. Última visita: 11-03-2014.

[34] IBM. AlphaWorks BPWS4J. http://www.alphaworks.ibm.com/tech/bpws4j, 2014. Última visita: 11-03-2014.

[35] Intalio. Intalio BPMS. http://bpms.intalio.com, 2013. Última visita: 11-03-2014.

[36] JBoss Community. jBPM. http://www.jboss.org/jbpm, 2014. Última visita: 05-02-2014.

[37] Oracle. Oracle BPEL Process Manager. http://www.oracle.com/technetwork/middleware/bpel/overview/index.html, 2012. Última visita: 11-03-2014.

[38] Orchestra Community. OW2 Orchestra. http://orchestra.ow2.org/xwiki/bin/view/Main/WebHome, 2013. Última visita: 11-03-2014.

[39] JBoss Community. Riftsaw. http://www.jboss.org/riftsaw, 2012. Última visita: 11-03-2014.

[40] NetWeaver. SAP Exchange Infrastructure. http://sap-press.de/download/dateien/751/sap_press_exchange_infra_engl.pdf, 2005. Última visita: 11-03-2014.

[41] OpenLink Virtuoso. Virtuoso Universal Server. http://docs.openlinksw.com/virtuoso/, 2014. Última visita: 11-03-2014.

[42] WSO2. Business Process Server. http://wso2.com/products/business-process-server/, 2014. Última visita: 11-03-2014.

[43] The Apache Software Foundation. http://www.apache.org/, 2014. Última visita: 11-03-2014.

[44] Apache Software Foundation. Apache Tomcat. `http://tomcat.apache.org/`, 2014. Última visita: 11-03-2014.

[45] Apache Software Foundation. Axis 2. `http://axis.apache.org/axis2/java/core/`, 2012. Última visita: 11-03-2014.

[46] The Eclipse Foundation. Eclipse. `http://www.eclipse.org/`, 2014. Última visita: 11-03-2014.

[47] Creative Commons Attribution 3.0 Unported license. Jenkins. `http://jenkins-ci.org/`, 2014. Última visita: 11-03-2014.

[48] The Apache Software Foundation. http://subversion.apache.org/. `http://www.eclipse.org/`, 2000. Última visita: 05-02-2014.

[49] B. Collins-Sussman, B. W. Fitzpatrick y C. Michael Pilato. *Version Control with Subversion*. O'Reilly Media, 2008. ISBN 978-0-596-51033-6.

[50] M. A. Pérez-Montero. Generador de casos de prueba aleatorio basado en especificaciones abstractas. `http://rodin.uca.es/xmlui/handle/10498/14661`, 2012. Última visita: 11-03-2014.

[51] Snakeyaml. Documentación oficial de SnakeYAML. `http://code.google.com/p/snakeyaml/wiki/Documentation`, 2014. Última visita: 11-03-2014.